



**ITS**

Institut  
Teknologi  
Sepuluh Nopember

**TUGAS AKHIR - TE 1599**

## **PENGUKURAN PERFORMANSI MESIN VIRTUAL PADA KOMPUTASI AWAN**

Nur Rohman Widiyanto  
NRP 2209100025

Dosen Pembimbing  
Mochamad Hariadi, ST., M.Sc., Ph.D.  
Christyowidiasmoro, ST., MT

JURUSAN TEKNIK ELEKTRO  
Fakultas Teknologi Industri  
Institut Teknologi Sepuluh Nopember  
Surabaya 2015



**ITS**  
Institut  
Teknologi  
Sepuluh Nopember

**TUGAS AKHIR - TE 1599**

## **PENGUKURAN PERFORMANSI MESIN VIRTUAL PADA KOMPUTASI AWAN**

Nur Rohman Widiyanto  
NRP 2209100025

Dosen Pembimbing  
Mochamad Hariadi, ST., M.Sc., Ph.D.  
Christyowidiasmoro, ST., MT

JURUSAN TEKNIK ELEKTRO  
Fakultas Teknologi Industri  
Institut Teknologi Sepuluh Nopember  
Surabaya 2015



**ITS**

Institut  
Teknologi  
Sepuluh Nopember

**FINAL PROJECT - TE 1599**

## **VIRTUAL MACHINE PERFORMANCE MEASUREMENT ON CLOUD COMPUTING**

Nur Rohman Widiyanto  
NRP 2209100025

Advisor  
Mochamad Hariadi, ST., M.Sc., Ph.D.  
Christyowidiasmoro, ST., MT

Departement of Electrical Engineering  
Faculty of Industrial Technology  
Sepuluh Nopember Institute of Technology  
Surabaya 2015

**PENGUKURAN PERFORMANSI MESIN VIRTUAL PADA  
KOMPUTASI AWAN**

**TUGAS AKHIR**

**Diajukan untuk Memenuhi Sebagian Persyaratan  
Untuk Memperoleh Gelar Sarjana Teknik  
Pada  
Bidang Studi Teknik Komputer dan Telematika  
Jurusan Teknik Elektro  
Institut Teknologi Sepuluh Nopember**

**Menyetujui :**

**Dosen Pembimbing I**

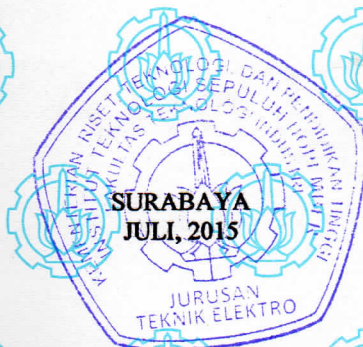


**Mochamad Hariadi, ST., M.Sc., Ph.D.**  
**NIP. 196912091997031002**

**Dosen Pembimbing II**



**Christyowidiasmoro, ST., MT.**  
**NIP. 198301272009121004**





# ABSTRAK

Nama Mahasiswa : Nur Rohman Widiyanto  
Judul Tugas Akhir : Pengukuran Performansi Mesin Virtual pada Komputasi Awan  
Pembimbing : 1. Mochamad Hariadi, ST., M.Sc., Ph.D.  
2. Christyowidiasmoro, ST., MT.

Berawal kebutuhan komputasi yang semakin beragam saat ini, maka dibangunlah komputasi awan untuk memenuhi kebutuhan tersebut. Komputasi awan dengan salah satu jenis layanannya yaitu IaaS (*Infrastructure as a Service*) memiliki keuntungan tidak perlu membeli komputer fisik, dan konfigurasi mesin virtual yang dapat dirubah dengan mudah. Di lakukan juga pengukuran performa dari mesin virtual yang merupakan layanan dari komputasi awan dengan spesifikasi yang beragam, selain itu performa dari mesin virtual juga dibandingkan dengan sebuah komputer dengan spesifikasi yang sama. Hal tersebut dilakukan guna memperoleh referensi berupa perbandingan performa dari mesin virtual dengan spesifikasi yang beragam, selain itu diperolehnya perbandingan performa antara mesin virtual yang merupakan layanan dari komputasi awan dengan sebuah komputer dengan spesifikasi yang sama. Komputasi awan dibangun menggunakan OpenStack sebagai sistem operasi utamanya. Pengguna juga dapat dengan mudah memilih sistem operasi yang dibutuhkan, mengatur spesifikasi sesuai dengan kebutuhan dan menjalankannya melalui antarmuka *website*. Pengukuran performa mesin virtual menggunakan Phoronix, sebuah aplikasi *open source* untuk *benchmark*. Phoronix memiliki sekitar 130 tes, seperti C-Ray yang digunakan untuk mengukur beban kerja pada *processor*. Hasil pengukuran tersebut menunjukkan bahwa performa dari mesin virtual tidak kalah dengan performa komputer. Setiap peningkatan 1 *core* pada CPU dapat berpengaruh pada berkurangnya waktu pemrosesan beban sebesar 18 s yang diukur dengan C-Ray.

Kata Kunci : IaaS (*Infrastructure as a Service*), Komputasi Awan, Mesin Virtual, OpenStack, Performa

# ABSTRACT

Name : Nur Rohman Widiyanto  
Title : *Virtual Machine Performance Measurement on Cloud Computing*  
Advisors : 1. Mochamad Hariadi, ST., M.Sc., Ph.D.  
2. Christyowidiasmoro, ST., MT.

*Beginning with diverse need of computation, the cloud computing was built to fulfill those needs. Cloud computing with one of its service type named IaaS (Infrastructure as a Service) has the advantage of not needing a physical computer, and easily changeable configuration of virtual machine. performance measuring is also done in the form of cloud computing with various specifications, moreover virtual machine's performance is compared with a computer having similar specifications. It is done in order to obtain references, in the form of performance comparison between virtual machine, which is a cloud computing service, with the Computer of similar specifications. Cloud computing is built with OpenStack as its main Operating System. Users can also easily choose the operating system needed, configure the needed specifications, and run it through the website interface. Virtual Machine's performance is measured using Phoronix, an open source application for benchmark. Phoronix owns 130 tests, like C-Ray which can be used to measure the workload in the processor. The measurement result show that the virtual machine performance is on par with Computer. Every 1 core in CPU affects in the reduced processing time amounting to 18 s which is measured using C-Ray.*

*Keywords : Cloud computing, IaaS (Infrastructure as a Service), Virtual machine, OpenStack, performance*

# KATA PENGANTAR

Puji dan syukur kehadiran Allah SWT atas segala limpahan berkah, rahmat, serta hidayah-Nya, penulis dapat menyelesaikan penelitian ini dengan judul **Analisa Performansi Mesin Virtual pada Komputasi Awan**.

Penelitian ini disusun dalam rangka pemenuhan bidang riset di Jurusan Teknik Elektro ITS, Bidang Studi Teknik Komputer dan Telematika, serta digunakan sebagai persyaratan menyelesaikan pendidikan S1. Penelitian ini dapat terselesaikan tidak lepas dari bantuan berbagai pihak. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Keluarga, Ibu, Bapak dan Saudara tercinta yang telah memberikan dorongan spiritual dan material dalam penyelesaian buku penelitian ini.
2. Bapak Dr. Tri Arief Sardjono, ST., MT. selaku Ketua Jurusan Teknik Elektro, Fakultas Teknologi Industri, Institut Teknologi Sepuluh Nopember.
3. Secara khusus penulis mengucapkan terima kasih yang sebesar-besarnya kepada Bapak Mochamad Hariadi, ST., M.Sc., Ph.D. dan Bapak Christyowidiasmoro, ST., MT. atas bimbingan selama mengerjakan penelitian.
4. Bapak-ibu dosen pengajar Bidang Studi Teknik Komputer dan Telematika, atas pengajaran, bimbingan, serta perhatian yang diberikan kepada penulis selama ini.
5. Seluruh teman-teman *B201-crew* Laboratorium Bidang Studi Teknik Komputer dan Telematika.

Kesempurnaan hanya milik Allah SWT, untuk itu penulis memohon segenap kritik dan saran yang membangun. Semoga penelitian ini dapat memberikan manfaat bagi kita semua. Amin.

Surabaya, Juni 2015

Penulis

# DAFTAR ISI

<b>Abstrak</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>KATA PENGANTAR</b>	<b>v</b>
<b>DAFTAR ISI</b>	<b>vii</b>
<b>DAFTAR GAMBAR</b>	<b>ix</b>
<b>DAFTAR TABEL</b>	<b>xi</b>
<b>DAFTAR KODE</b>	<b>xiii</b>
<b>1 PENDAHULUAN</b>	<b>1</b>
1.1 Latar belakang . . . . .	1
1.2 Permasalahan . . . . .	2
1.3 Tujuan . . . . .	2
1.4 Batasan masalah . . . . .	2
1.5 Sistematika Penulisan . . . . .	2
1.6 Relevansi . . . . .	3
<b>2 TINJAUAN PUSTAKA</b>	<b>5</b>
2.1 OpenStack . . . . .	5
2.2 Virtualisasi . . . . .	7
2.3 Phoronix Test Suite . . . . .	8
<b>3 DESAIN DAN IMPLEMENTASI SISTEM</b>	<b>11</b>
3.1 Desain Sistem . . . . .	11
3.2 Desain Jaringan . . . . .	12
3.3 Alur Implementasi Sistem . . . . .	14
3.4 Perangkat Keras dan Sistem Operasi . . . . .	14
3.5 Pembuatan Controller Node . . . . .	16
3.5.1 Keystone (Identity Service) . . . . .	17
3.5.2 Glance (Image Service) . . . . .	19
3.5.3 Nova (Compute) pada Controller . . . . .	21



3.5.4	Cinder (Block Storage Service) . . . . .	23
3.6	Pembuatan Compute Node . . . . .	25
3.7	Pembuatan Horizon (Dashboard) . . . . .	27
<b>4</b>	<b>PENGUJIAN DAN ANALISA</b>	<b>29</b>
4.1	Pengujian Layanan . . . . .	29
4.1.1	Uji Coba Keystone (Identity Service) . . . . .	29
4.1.2	Uji Coba Glance (Image Service) . . . . .	34
4.1.3	Uji Coba Nova (Compute Service) . . . . .	37
4.1.4	Uji Coba Mesin Virtual (Instance) . . . . .	38
4.1.5	Uji Coba Cider (Block Storage Service) . . . . .	46
4.2	Pengujian Topologi Jaringan . . . . .	50
4.3	Kemampuan Maksimal Komputasi Awan . . . . .	50
4.4	Perbandingan Performa Mesin Virtual . . . . .	51
4.4.1	Performa Komputasi (CPU) . . . . .	53
4.4.2	Performa Penyimpanan . . . . .	62
4.4.3	Performa Memori (RAM) . . . . .	65
4.4.4	Apache Benchmark . . . . .	72
<b>5</b>	<b>PENUTUP</b>	<b>75</b>
5.1	Kesimpulan . . . . .	75
5.2	Saran . . . . .	76
	<b>DAFTAR PUSTAKA</b>	<b>77</b>
	<b>Biografi Penulis</b>	<b>79</b>

## DAFTAR TABEL

3.1	Spesifikasi dari <i>controller</i> dan <i>compute node</i> . . . . .	15
3.2	Spesifikasi dari <i>storage node</i> . . . . .	15
4.1	Jumlah spesifikasi dari empat <i>compute node</i> . . . . .	51
4.2	Spesifikasi (flavor) dari mesin virtual . . . . .	52
4.3	Spesifikasi dari komputer yang akan dibandingkan . . . . .	52
4.4	<i>Flavor</i> mesin virtual untuk mengukur performa komputasi. . . . .	53
4.5	<i>Flavor</i> mesin virtual untuk mengukur performa memori. . . . .	53
4.6	Waktu pemrosesan beban kerja antara mesin virtual dan komputer . . . . .	54
4.7	Waktu pemrosesan beban kerja pada mesin virtual . . . . .	56
4.8	Hasil pengujian memori <i>bandwidth</i> pada komputer dan mesin virtual . . . . .	57
4.9	Hasil pengujian performa memori <i>bandwidth</i> pada mesin virtual . . . . .	58
4.10	Hasil pengujian kecepatan proses encoding . . . . .	60
4.11	Waktu yang diperlukan saat <i>encoding</i> . . . . .	61
4.12	Hasil pengujian penyimpanan dengan Aio-Stress . . . . .	63
4.13	Hasil pengujian penyimpanan dengan FIO . . . . .	64
4.14	Hasil pengujian <i>throughput</i> memori saat mengolah data <i>integer</i> pada mesin virtual dan komputer . . . . .	66
4.15	Hasil pengujian <i>throughput</i> memori saat mengolah <i>floating-point</i> pada mesin virtual dan komputer . . . . .	66
4.16	Hasil pengujian <i>throughput</i> saat memproses data <i>integer</i> pada mesin virtual . . . . .	67
4.17	Hasil pengujian <i>throughput</i> saat memproses <i>floating-point</i> pada mesin virtual . . . . .	67
4.18	Hasil perbandingan kecepatan <i>cache</i> memori antara mesin virtual dengan komputer . . . . .	70
4.19	Hasil perbandingan <i>bandwidth cache</i> memori pada mesin virtual . . . . .	71
4.20	Hasil pengujian mesin virtual dan komputer sebagai <i>webserver</i> dengan Apache Benchmark . . . . .	72

# DAFTAR GAMBAR

2.1	Konsep arsitektur dasar OpenStack . . . . .	5
2.2	Arsitektur inti OpenStack. . . . .	6
2.3	Metode virtualisasi . . . . .	7
3.1	Rancangan sistem layanan yang digunakan . . . . .	12
3.2	Desain jaringan . . . . .	13
3.3	Alur implementasi sistem . . . . .	14
3.4	Hubungan antar layanan pada OpenStack . . . . .	16
3.5	Skema interaksi antara pengguna dengan keystone . . . . .	19
4.1	<i>Login screen</i> . . . . .	30
4.2	Penambahan pengguna melalui antarmuka web . . . . .	31
4.3	Daftar pengguna dan layanan melalui <i>console</i> . . . . .	31
4.4	Daftar pengguna dan layanan melalui antarmuka . . . . .	32
4.5	Daftar <i>tenant</i> melalui <i>console</i> . . . . .	33
4.6	Pengguna dan layanan saat pembuatan <i>project</i> . . . . .	33
4.7	Daftar pengguna dan <i>role</i> melalui <i>console</i> . . . . .	34
4.8	Penambahan <i>image</i> melalui antarmuka . . . . .	35
4.9	Daftar <i>image</i> yang berhasil terunggah dilihat melalui console . . . . .	36
4.10	Daftar layanan dari nova yang sedang berjalan dilihat melalui console . . . . .	36
4.11	Daftar layanan dari nova yang sedang berjalan, bila dilihat melalui antarmuka <i>web</i> . . . . .	37
4.12	Daftar <i>image</i> sistem operasi dari glance yang terbaca oleh nova . . . . .	37
4.13	Daftar <i>key pairs</i> melalui <i>console</i> . . . . .	38
4.14	Pembuatan <i>keypair</i> melalui antarmuka . . . . .	39
4.15	Daftar <i>flavor</i> melalui <i>console</i> . . . . .	39
4.16	Daftar <i>image</i> melalui antarmuka web . . . . .	40
4.17	Daftar <i>nova-network</i> melalui <i>console</i> . . . . .	40
4.18	Daftar secgroup melalui <i>console</i> . . . . .	41
4.19	Pembuatan mesin virtual melalui antarmuka . . . . .	42
4.20	Penambahan <i>flavor</i> melalui antarmuka . . . . .	42
4.21	Tampilan monitoring <i>hypervisors</i> . . . . .	43

4.22	Perintah tampilkan alamat VNC yang berupa URL .	43
4.23	Akses mesin virtual melalui <i>browser</i> . . . . .	44
4.24	Pengaturan <i>security group</i> melalui antarmuka . . . .	45
4.25	Penambahan <i>security group</i> melalui antarmuka . . . .	46
4.26	Melihat daftar mesin melalui <i>console</i> . . . . .	47
4.27	Layanan-layanan pembangun cinder . . . . .	47
4.28	Penambahan <i>volume</i> melalui antarmuka . . . . .	48
4.29	Penambahan penyimpanan melalui antarmuka . . . .	49
4.30	Desain jaringan tanpa jaringan eksternal . . . . .	50
4.31	Perbedaan waktu pemrosesan beban kerja antara me- sin virtual dengan komputer . . . . .	54
4.32	Perbedaan waktu pemrosesan beban kerja pada me- sin virtual . . . . .	56
4.33	Perbandingan memori bandwidth antara mesin vir- tual dengan komputer . . . . .	58
4.34	Perbedaan memori bandwith pada mesin virtual . .	59
4.35	Perbandingan kecepatan encoding . . . . .	60
4.36	Perbedaan waktu saat encoding . . . . .	61
4.37	Perbandingan bandwidth penyimpanan . . . . .	63
4.38	Perbandingan I/O penyimpanan dengan FIO . . . .	64
4.39	Perbandingan <i>throughput</i> saat memproses <i>integer</i> pa- da komputer dan mesin virtual . . . . .	66
4.40	Perbandingan <i>throughput</i> saat memproses <i>floating po- int</i> antara mesin virtual dengan komputer . . . . .	68
4.41	Perbedaan <i>throughput</i> saat memproses data <i>integer</i> pada mesin virtual. . . . .	68
4.42	Perbedaan <i>throughput</i> saat memproses <i>floating-point</i> pada mesin virtual . . . . .	69
4.43	Perbandingan kecepatan <i>cache</i> memori antara mesin virtual dengan komputer . . . . .	70
4.44	Perbedaan <i>bandwith cache</i> memori mesin virtual . .	71
4.45	Kemampuan menerima permintaan setiap detiknya .	73



## DAFTAR KODE

3.1	Galat saat pemasangan glance . . . . .	20
3.2	Hubungan antara database, keystone dan glance . . .	20
3.3	Hubungan database, keystone, glance dan nova . . .	22
3.4	Hubungan antara database, keystone, dan cinder . . .	24
3.5	Hubungan nova pada <i>compute node</i> dengan <i>controller node</i> . . . . .	26
3.6	Nova-network pada <i>controller node</i> . . . . .	27
3.7	Nova-network pada <i>compute node</i> . . . . .	27
3.8	Penghubung <i>controller node</i> dengan antarmuka <i>web</i> . . .	28
4.1	Login script pada keystone dengan nama pengguna admin. . . . .	30
4.2	Perintah untuk tambah pengguna . . . . .	31
4.3	Perintah untuk tambah <i>tenant</i> atau <i>project</i> . . . . .	32
4.4	Perintah untuk memberi <i>role</i> ke sebuah <i>project</i> . . . .	33
4.5	Perintah untuk mengunggah <i>image</i> ke glance . . . . .	34
4.6	Perintah untuk menambahkan <i>keypair</i> ke nova . . . .	38
4.7	Perintah untuk membuat mesin virtual ( <i>instance</i> ) melalui <i>console</i> . . . . .	41
4.8	Perintah untuk memberikan izin agar mesin virtual dapat diakses melalui SSH . . . . .	45
4.9	Perintah untuk memberikan izin agar mesin virtual dapat menerima paket ICMP . . . . .	45
4.10	Perintah untuk menambah <i>volume</i> melalui <i>console</i> . .	48
4.11	Perintah untuk menambah <i>volume</i> ke mesin virtual melalui <i>console</i> . . . . .	49

## BIOGRAFI PENULIS



Nur Rohman Widiyanto, lahir pada 26 Maret 1991 di Lamongan, Jawa Timur. Penulis lulus dari SMP Negeri 1 Paciran pada tahun 2006 kemudian melanjutkan pendidikan ke SMA Muhammadiyah 1 Gresik hingga akhirnya lulus pada tahun 2009. Penulis kemudian melanjutkan pendidikan Strata satu ke Jurusan Teknik Elektro ITS Surabaya bidang studi Teknik Komputer dan Telematika. Saat di kuliah penulis aktif menjadi staff PSDM (Pemberdayaan Sumber Daya Mahasiswa) BEM ITS 2010/2011. Penulis juga aktif menjadi Asisten laboratorium B201 (Telematika) hingga saat ini dan pernah menjabat sebagai koordinator asisten Lab B201 periode 2012/2013. Selama masa kuliah penulis aktif dalam mengikuti ajang perlombaan seperti PKM (Program Kreativitas Mahasiswa), aktif dalam *development group networking* dan juga sebagai *administrator* jaringan Lab B201. Penulis sangat tertarik dengan segala hal yang berhubungan dengan komputer, dan berencana mendalami cabang ilmu komputer lain selain jaringan komputer.

# BAB 1

## PENDAHULUAN

Penelitian ini di latar belakang oleh berbagai kondisi yang menjadi acuan. Selain itu juga terdapat beberapa permasalahan yang akan dijawab sebagai luaran dari penelitian.

### 1.1 Latar belakang

Kebutuhan akan komputasi yang semakin meningkat, seperti pada level pengguna atau *server*. Pada komputasi awan (*cloud computing*) pengguna dapat memperoleh berbagai layanan yang berupa komputasi (CPU), memori, penyimpanan data dan lain sebagainya. Semuanya disediakan sebagai sebuah layanan oleh pihak ketiga [1]. Jenis dari layanan itu adalah IaaS (*Infrastructure as a Service*) yang memiliki keuntungan diantaranya pengguna tidak perlu membeli komputer fisik, melainkan hanya cukup melakukan konfigurasi pada mesin virtual dengan mudah [2]. Konfigurasi mesin virtual dapat dirubah dengan mudah, contohnya saat komputer virtual tersebut mengalami kelebihan beban maka dapat ditambahkan CPU, RAM, media penyimpanan dan sebagainya dengan segera.

Keuntungan lainnya adalah pengurangan biaya investasi perangkat keras, kemudahan *backup* serta *recovery*, pengurangan pemanasan data center, pengurangan biaya sewa slot server, kemudahan skalabilitas dan pengelolaan [3]. Di sisi lain muncul beberapa pertanyaan bagaimana performa sistem operasi yang dipasang pada komputer atau mesin virtual, dan bagaimana perbandingan performa dari mesin virtual tersebut dengan sistem operasi yang dipasang sebuah komputer dengan spesifikasi yang sama.

OpenStack berperan sebagai sistem operasi pada komputasi awan, fitur yang dimiliki mendukung pembuatan dan penyediaan IaaS. Bentuk layanan yang dibuat dalam penelitian ini berupa mesin virtual yang kemudian diukur performanya. Selain itu juga diukur performa dari sebuah komputer, keduanya menggunakan sistem operasi dan spesifikasi yang sama. Hal ini dilakukan guna memperoleh hasil perbandingan performa antara mesin virtual dengan sebuah komputer dan juga mengukur performa mesin virtual dengan



spesifikasi yang terus dinaikan demi memperoleh referensi dalam pemilihan kebutuhan komputasi yang tepat.

## **1.2 Permasalahan**

Berawal dari permasalahan akan kebutuhan komputasi yang beragam dalam skala lab, maka dibangunlah komputasi awan untuk memenuhi kebutuhan tersebut. Layanan yang dibangun berupa IaaS (*Infrastructure as a Service*) dengan mesin virtual sebagai layanan utamanya. Di lakukan juga pengukuran performa dari mesin virtual yang merupakan layanan dari komputasi awan dengan spesifikasi yang beragam, selain itu performa dari mesin virtual juga dibandingkan dengan sebuah komputer menggunakan spesifikasi yang sama.

## **1.3 Tujuan**

Tujuan utama dari penelitian ini adalah membangun komputasi awan skala lab demi memenuhi kebutuhan komputasi yang beragam di dalamnya. Di perolehnya referensi berupa perbandingan performa dari mesin virtual dengan spesifikasi yang beragam, selain itu di dapat juga perbandingan performa antara mesin virtual yang merupakan layanan dari komputasi awan dengan sebuah komputer dengan spesifikasi yang sama. Manfaat lain dari penelitian ini adalah diperolehnya referensi untuk mengatasi peningkatan kebutuhan komputasi yang beragam dengan cara penggunaan yang mudah.

## **1.4 Batasan masalah**

Batasan masalah yang timbul dari permasalahan Tugas Akhir ini adalah :

1. Layanan komputasi awan yang dibangun berupa IaaS (*Infrastructure as a Service*) dengan menyediakan mesin virtual skala lab.

## **1.5 Sistematika Penulisan**

Laporan penelitian Tugas akhir ini tersusun dalam sistematika dan terstruktur sehingga mudah dipahami dan dipelajari oleh pembaca maupun seseorang yang ingin melanjutkan penelitian ini. Alur sistematika penulisan laporan penelitian ini yaitu :

## 1. BAB I Pendahuluan

Bab ini berisi uraian tentang latar belakang permasalahan, penegasan dan alasan pemilihan judul, sistematika laporan, tujuan dan metodologi penelitian.

## 2. BAB II Dasar Teori

Pada bab ini berisi tentang uraian secara sistematis teori-teori yang berhubungan dengan permasalahan yang dibahas pada penelitian ini. Teori-teori ini digunakan sebagai dasar dalam penelitian, yaitu informasi terkait teknologi komputasi awan, mesin virtual, dan teori-teori penunjang lainnya.

## 3. BAB III Perancangan Sistem dan Impementasi

Bab ini berisi tentang penjelasan-penjelasan terkait sistem yang akan dibuat. Guna mendukung itu digunakanlah blok diagram atau *work flow* agar sistem yang akan dibuat dapat terlihat dan mudah dibaca untuk implentasi pada pelaksanaan tugas akhir.

## 4. BAB IV Pengujian dan Analisa

Bab ini menjelaskan tentang pengujian yang dilakukan terhadap sistem dalam penelitian ini dan menganalisa sistem. Spesifikasi perangkat keras dan perangkat lunak yang diuji juga disebutkan dalam bab ini. Sehingga ketika akan dikembangkan lebih jauh, spesifikasi perlengkapannya bisa dipenuhi dengan mudah tanpa harus melakukan ujicoba perangkat lunak maupun perangkat keras lagi.

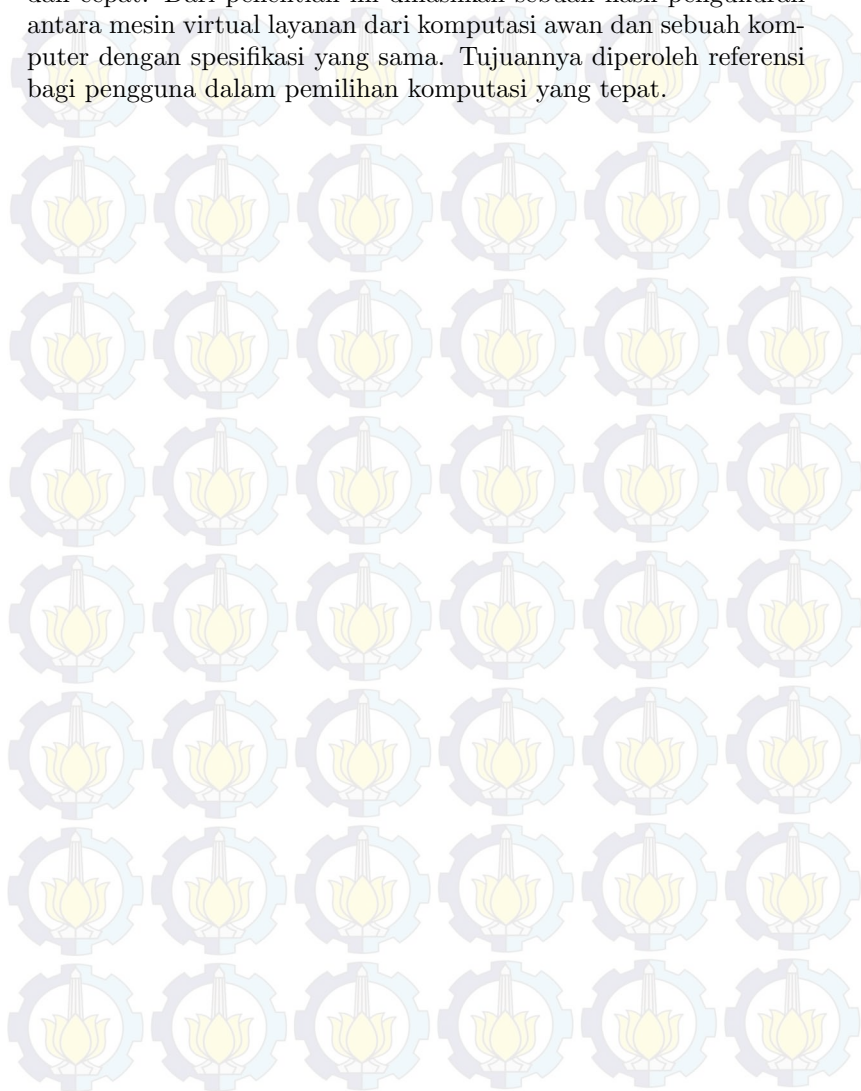
## 5. BAB V Penutup

Bab ini merupakan penutup yang berisi kesimpulan yang diambil dari penelitian dan pengujian yang telah dilakukan. Saran dan kritik yang membangun untuk mengembangkan lebih lanjut juga dituliskan pada bab ini.

# 1.6 Relevansi

Penelitian mengenai komputasi awan merupakan bidang penelitian yang sangat dibutuhkan dan dipakai dalam pemenuhan kebutuhan komputasi yang semakin beragam saat ini. Layanan *Infras-structure as a Service* (IaaS) dengan mesin virtual sebagai layanan utamanya. Di mana pengguna dapat mengatur kebutuhannya se-

perti memori, penyimpanan dan komputasi (CPU) dengan mudah dan cepat. Dari penelitian ini dihasilkan sebuah hasil pengukuran antara mesin virtual layanan dari komputasi awan dan sebuah komputer dengan spesifikasi yang sama. Tujuannya diperoleh referensi bagi pengguna dalam pemilihan komputasi yang tepat.



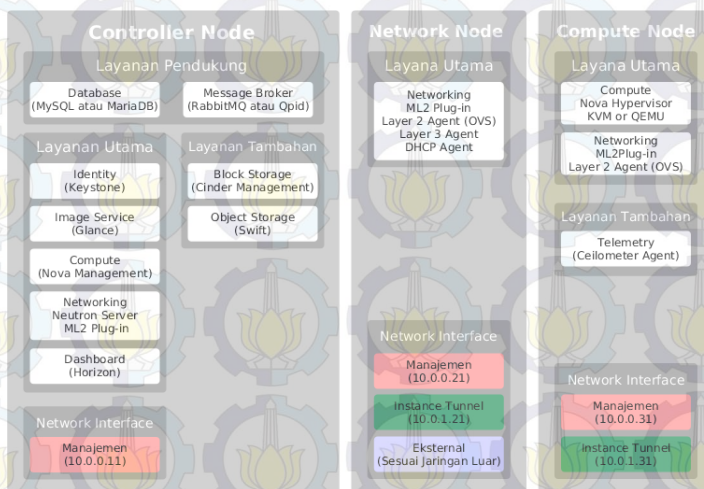
## BAB 2

### TINJAUAN PUSTAKA

Demi mendukung penelitian ini, dibutuhkan beberapa teori penunjang sebagai bahan acuan dan refrensi. Dengan demikian penelitian ini menjadi lebih terarah.

#### 2.1 OpenStack

OpenStack adalah sistem operasi pada komputasi awan yang mengontrol *big pool*, penyimpanan data, dan jaringan di seluruh data center, semua dikelola melalui *dashboard* yang memberikan kontrol *administrator* yang ditampilkan dan diakses melalui *browser* dengan skema seperti pada Gambar 2.1.

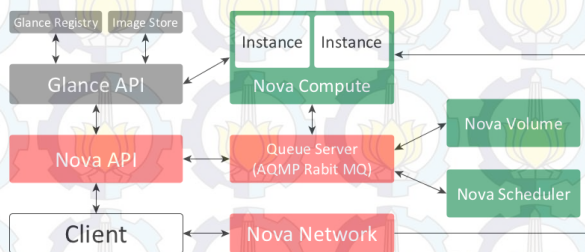


Gambar 2.1: Konsep arsitektur dasar OpenStack

1. OpenStack Networking atau Neutron adalah sebuah sistem untuk mengatur jaringan pada OpenStack, didalamnya terdapat API yang berfungsi untuk mengelola jaringan dan alamat IP (*internet protocol*) bagi penggunaannya [4].



2. Swift atau Object Storage merupakan media penyimpanan obyek pada OpenStack. Swift dilengkapi dengan *proxy server*, *object server*, sebuah *account server*, sebuah *container server* dan *ring*. Sebuah penyimpanan jangka panjang dengan data yang statis, dapat diambil dan diperbaharui. Fungsi utama dan fiturnya adalah sebagai media penyimpanan yang besar dan aman, mengurangi redudansi data, kemampuan arsip dan media *streaming*.
3. Nova atau OpenStack *Compute* merupakan bagian utama dari OpenStack, bertugas sebagai kontroler dari sistem komputasi awan. Memiliki enam komponen seperti Nova-API, *Message Queue* dengan Rabbitmq, Nova-Compute, Nova-Network, Nova-Volume dan Nova-Scheduler. Seluruh komponen pada arsitektur Nova mengikuti aturan *shared-nothing* dan *messaging-based*, maksud dari *shared-nothing* adalah setiap komponen dapat dipasang pada server manapun. Misalnya, *compute controller*, *volume controller*, *network controller* dan *object storage* dapat dipasang dalam satu server atau empat server secara terpisah. Seperti pada Gambar 2.2. *Queue Server* berada ditengah-tengah arsitekstur, maksud dari messaging-based adalah terjalin komunikasi pada setiap kontroler pada komputasi awan diantaranya *volume*, *network*, dan penjadwalan melalui *queue server* pada *advanced message queue protocol* (AMQP).



Gambar 2.2: Arsitektur inti OpenStack.

4. OpenStack *Image Service* dibuat untuk mencari dan mengambil *Image* dari mesin virtual. Fitur ini dinamakan Glance,

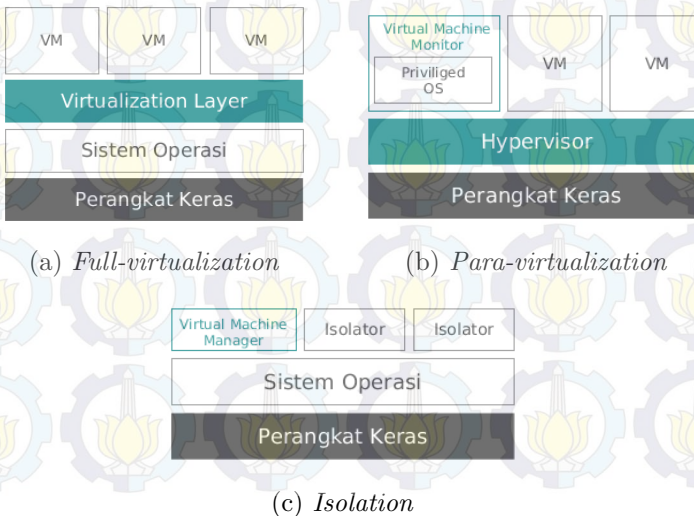


*Glance Registry* dan *Glance Control* merupakan bagian utamanya.

5. Keystone atau Identify Service adalah keamanan utama pada komputasi awan yang menyediakan layanan authentication dan authorization dari OpenStack [2].
6. Cinder atau *Block Storage* menyediakan blok penyimpanan atau volume dalam menjalankan layanan OpenStack. Dengan arsitektur yang mudah diatur setiap penambahan blok penyimpanan datanya.
7. Dashboard atau Horizon disediakan untuk administrator dan pengguna yang berupa tampilan antar muka untuk mengakses, mengatur secara langsung melalui browser pada layanan komputasi awan [4].

## 2.2 Virtualisasi

Virtualisasi (*Virtualization*) adalah cara yang memungkinkan sebuah komputer atau mesin fisik untuk menjalankan beberapa komputer secara virtual di atasnya dengan menggunakan batuan perangkat lunak.



**Gambar 2.3:** Metode virtualisasi

Tiga metode virtualisasi diantaranya *full-virtualization*, *para-virtualization* dan *OS-Level virtualization* (Isolasi). *Full-virtualization* menyediakan mesin virtual mirip atau menyesuaikan dengan spesifikasi perangkat keras pada komputer induk seperti pada Gambar 2.3 (a). Tipe ini menawarkan keamanan, kemudahan pemasangan dan pemindahan pada setiap mesin virtual. Contoh dari pengguna metode virtualisasi ini adalah produk-produk dari VMware, VirtualBox dan KVM.

Pada Gambar 2.3 (b), *para-virtualization* menyediakan mesin virtual yang lebih abstrak. Mesin virtual tersebut memiliki spesifikasi yang mirip tetapi tidak persis dengan perangkat keras dari komputer induknya. Xen dan HyperV menggunakan metode virtualisasi jenis ini.

Pembagian sumber daya antar pengguna mesin virtual yang diatur oleh *virtual machine manager* seperti pada Gambar 2.3 (c). Sejumlah mesin virtual berjalan diatas beberapa salinan kernel sistem operasi yang sama dengan komputer induk. Melalui proses tersebutlah maka terbentuk sebuah mesin virtual melalui sebuah kernel sistem operasi. Metode ini hanya terdapat di linux, contohnya OpenVZ.

*Isolasion* dinilai memiliki performa paling baik. namun kelemahannya metode tersebut hanya bisa digunakan pada Linux dan sitem operasi yang dilakukan harus sama dengan sistem operasi dari komputer induk. Sedangkan *para-virtualisasi* dibuat dengan memodifikasi kernel agar mampu melakukan virtualisasi. *Full-virtualization* hanya dapat dibangun dengan perangkat keras dan processor yang mendukung virtualisasi yang akan dimodifikasi kernelnya sebelum melakukan virtualisasi. Namun dalam mengakses perangkat keras dari komputer induk, mesin virtul harus melewati layer virtualisasi terlebih dahulu seperti pada Gambar 2.3 (a) [5].

## 2.3 Phoronix Test Suite

Phoronix Test Suite (versi 5.6.0; PTS) [28] adalah perangkat lunak open-source yang digunakan untuk melakukan pengujian *benchmark*, selain itu PTS juga dapat berjalan pada banyak sistem operasi (*multiplatform*) seperti Linux, Mac OSX, BSD dan Microsoft Windows. *Benchmark* sendiri adalah suatu metode untuk meli-

hat performa atau kemampuan komputer melalui tes, dari hasil tes tersebut kemudian dibandingkan dengan komputer lainnya. *Benchmarking* biasanya akan menghasilkan hasil akhir berupa angka atau skor. Dengan membandingkan skor dari hasil proses tersebut, maka akan terlihat komputer dengan performa yang lebih baik.

Lebih dari 130 tes profil dan 60 rangkaian pengujian pada Phoronix Test Suite dapat digunakan. Pengguna harus melakukan download setiap rangkaian atau profilnya saat pertama kali melakukan *benchmarking*, tetapi untuk pemakaian selanjutnya tidak perlu melakukan download. Mulai dari kemampuan komputasi (CPU), memori, proses penyimpanan (I/O) dan pengolahan grafis pada komputer, perangkat bergerak dan komputasi awan. Pengguna tidak perlu menjalankan 130 tes profil untuk melakukan pengujian, cukup memilih beberapa rangkaian atau tes yang dibutuhkan [6].

## BAB 3

# DESAIN DAN IMPLEMENTASI SISTEM

Penelitian ini dilaksanakan sesuai dengan desain sistem berikut dengan implementasinya. Desain sistem merupakan konsep dari pembuatan dan perancangan infrastruktur dan kemudian diwujudkan dalam bentuk blok-blok alur yang harus dikerjakan. Pada bagian implementasi merupakan pelaksanaan teknis untuk setiap blok pada desain sistem.

### 3.1 Desain Sistem

Penelitian ini bertujuan untuk mengukur performa mesin virtual yang merupakan layanan dari komputasi awan dan sebuah komputer. Spesifikasi dan sistem operasi yang sama bagi keduanya adalah sebuah bagian yang wajib. Sebelum melakukan pengukuran performa, infrastruktur dan sistem dari komputasi awan harus dibangun terlebih dahulu.

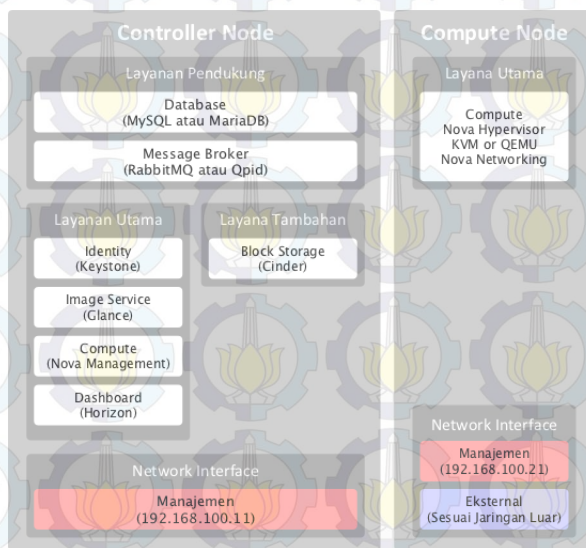
Pembangun infrastruktur komputasi awan, perangkat lunak yang digunakan adalah OpenStack. Fungsi utamanya adalah untuk melakukan kontrol dan menyediakan layanan mesin virtual. Bagian terpenting dari OpenStack adalah *controller* dan *compute node*. *Controller node* menjalankan keystone (*identity service*), glance (*image service*), nova-controller sebagai pengontrol nova-compute beserta nova-network pada setiap *compute node*, cinder (*block storage service*), dan *dashboard* untuk tampilan yang dapat diakses melalui *browser*. Selain itu beberapa layanan pendukung pada *controller node* diatarannya MariaDB untuk *database*, *message broker* untuk komunikasi antar layanan dengan menggunakan RabbitMQ, dan NTP (*Network Time Protocol*) yang bertugas untuk sinkronisasi waktu antar komputer.

Pada *compute node* berfungsi untuk menjalankan hypervisors, di sana merupakan tempat mesin virtual dioperasikan. Jumlah dari *compute node* dapat terus ditambah sesuai kebutuhan mesin virtual yang ingin dipakai. Sedangkan untuk Neuron (*networking node*)



berperan untuk menjalankan layanan pada layer 2 TCP/IP, pengaturan jaringan luar, sebagian layer 3 seperti NAT (*Network Address Translation*) dan DHCP (*Dynamic Host Configuration Protocol*). Namun *networking node* tidak digunakan dalam penelitian ini, karena tersedianya *gateway* yang mengatur jaringan.

Rancangan dasar dari OpenStack seperti pada Gambar 3.1 terdiri dari dua *node* dan terdapat berbagai layanan penyusun dan layanan tambahan di dalamnya. Layanan tambahan seperti Swift (*Object Storage*), Trove (*Database service*), Heat (*Orchestration*), dan ceilometer (*Telemetry*) tidak digunakan dalam penelitian ini karena masih belum diperlukan. Sedangkan layanan tambahan yang digunakan adalah cinder (*Block storage*). Jumlah dari *controller node* dan *compute node* bisa terus ditambah sesuai kebutuhan.

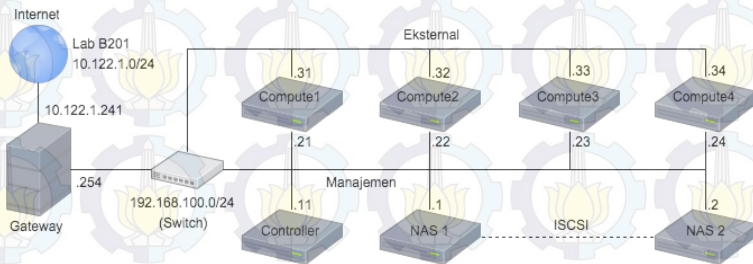


**Gambar 3.1:** Rancangan sistem layanan yang digunakan

## 3.2 Desain Jaringan

Desain jaringan pada penelitian ini menggunakan jaringan privat pada laboratorium Telematika (B201). Alamat IP (*Internet*

*Protocol*) dari *gateway* yang digunakan untuk mengakses jaringan luar adalah 10.122.1.241, berada dalam jaringan 10.122.1.0/24 milik laboratorium Telematika (B201). Sedangkan IP alias yang digunakan untuk implementasi komputasi awan adalah 192.168.100.0/24 dengan alamat IP *gateway* 192.168.100.254 seperti pada Gambar 3.2.



**Gambar 3.2:** Desain jaringan

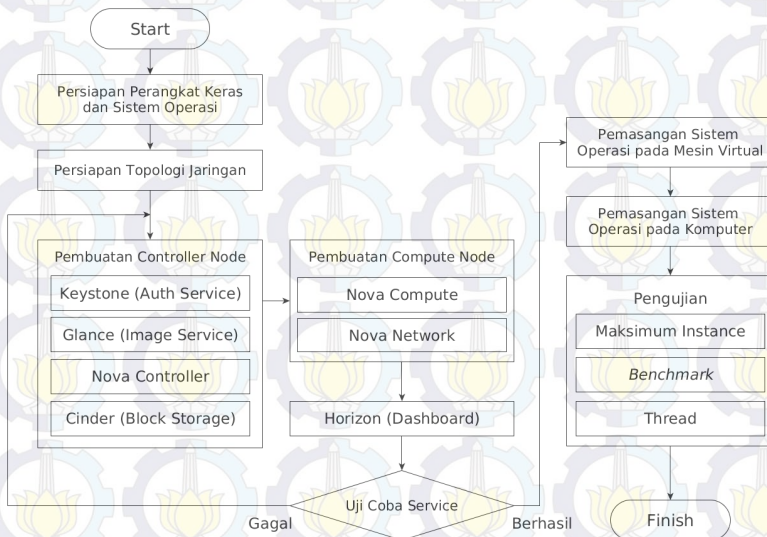
Seperti yang terlihat pada Gambar 3.2, tiga unit komputer yang terhubung dengan *gateway* melalui *switch*. Di antaranya satu *controller node* dan dua *compute node*. Controller terhubung dengan jaringan manajemen, begitu juga compute1, compute2, compute3 dan compute4. Fungsi utama dari jaringan manajemen adalah untuk sinkronisasi dan komunikasi antar *node*, Controller berperan sebagai pengatur keseluruhan sistem komputasi awan. Kemudian compute1, compute2, compute3 dan compute4 terhubung dengan jaringan eksternal. Hal ini bertujuan untuk memberikan alokasi alamat IP pada mesin virtual yang merupakan layanan dari sistem komputasi awan.

Jaringan manajemen dan eksternal pada umumnya dibuat berberda jaringan, misalkan untuk jaringan manajemen hanya menggunakan IP privat atau lokal sedangkan jaringan eksternal serharusnya menggunakan IP publik atau IP yang sengaja dibuat untuk diakses oleh banyak pengguna. Pada penelitian ini alamat IP sengaja dibuat sama, karena gateway dari komputasi awan sudah terdaftar dalam routing table gateway laboratorium Telematika. Hal ini bertujuan untuk membatasi pengguna komputasi awan hanya sebatas laboratorium Telematika.

Terdapat dua *storage node* dengan nama NAS1 dan NAS2, keduanya saling terhubung melalui jaringan manajemen. Tugas utama dari keduanya adalah sebagai NAS (*Network-attached Storage*) yang menyediakan penyimpanan dan terhubung dengan cinder melalui jaringan manajemen. Media penyimpanan pada NAS2 digabungkan menjadi satu melalui jaringan dengan NAS1 menggunakan ISCSI (*Internet Small Computer System Interface*).

### 3.3 Alur Implementasi Sistem

Alur kerja dalam pengerjaan tugas akhir ini terbagi mejadi tujuh tahapan proses dan satu percabangan. Dalam masing-masing proses memiliki luaran yang dihasilkan dan menjadi input dari proses selanjutnya seperti yang terlihat pada Gambar 3.3.



Gambar 3.3: Alur implementasi sistem

### 3.4 Perangkat Keras dan Sistem Operasi

Komputer yang dipakai dalam penelitian ini sejumlah tujuh unit, satu komputer berperan sebagai controller node dan empat komputer lainnya sebagai compute node. Komputer yang digunak-

an sebagai *controller* dan *compute* memiliki spesifikasi yang sama seperti pada Tabel 3.1. Sedangkan untuk dua unit lainnya merupakan NAS (*Network Attached Storage*) dengan spesifikasi yang sama namun berbeda pada jumlah Disk Storage seperti pada Tabel 3.2.

**Tabel 3.1:** Spesifikasi dari *controller* dan *compute node*

Spesifikasi	Keterangan
Jumlah Prosesor	2
Jumlah Core	Quad-core
Kecepatan Prosesor	3.46 GHz
Jenis Prosesor	Intel® Xeon® 5600 series
<i>Cache</i>	8 MB L3
Arsitektur	x86_64
Memori	6 GB
<i>Disk Storage</i>	300 GB
<i>Power Supplay</i>	460 Watt

**Tabel 3.2:** Spesifikasi dari *storage node*

Spesifikasi	Keterangan
Jumlah Prosesor	2
Jumlah Core	Quad-core
Kecepatan Prosesor	3.46 GHz
Jenis Prosesor	Intel® Xeon® 5600 series
<i>Cache</i>	8 MB L3
Arsitektur	x86_64
Memori	6 GB
<i>Disk Storage</i>	
- NAS1	6927 GB
- NAS2	7927 GB
<i>Power Supplay</i>	460 Watt

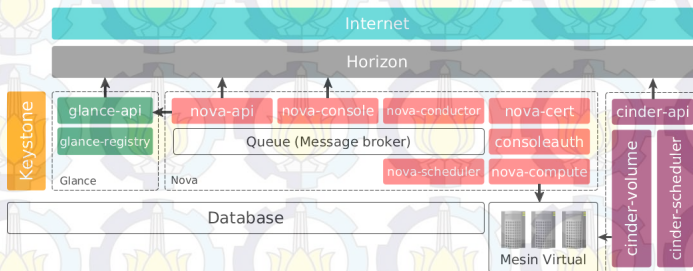


Spesifikasi minimum dari OpenStack berbeda-beda setiap node-nya, untuk *controller node* minimum 1 prosesor, 2 GB memori dan 5 GB *disk storage*. Kemudian pada *compute node* adalah 1 prosesor, 2 GB memori dan 10 GB *disk storage*, sedangkan pada *networking node* adalah 1 prosesor, 2 GB memori dan 5 GB *disk storage*. OpenStack mendukung beberapa sistem operasi linux di antaranya Suse Linux Enterprise Server dan OpenSuse, RedHat, Fedora, CentOS, Ubuntu dan Debian [4].

Sistem operasi yang akan digunakan dalam penelitian ini adalah Ubuntu 14.04 (Trusty Tahr) dengan arsitektur 64-bit pada *controller node* dan *compute node*. Versi dari OpenStack yang digunakan adalah Juno yang rilis pada bulan Oktober 2014. Sedangkan pada *storage node* sistem operasi yang digunakan adalah Windows Server Storage 2008 R2 dengan Arsitektur 64-bit.

### 3.5 Pembuatan Controller Node

Pemasangan perangkat lunak pada komputer yang akan berperan sebagai *controller node*, terdiri dari beberapa layanan yang saling terhubung diantaranya MariaDB untuk manajemen *database*, rabbitmq, glance, keystone, nova dan horizon.



Gambar 3.4: Hubungan antar layanan pada OpenStack

Beberapa layanan seperti keystone, glance dan nova terbentuk dari beberapa dependence atau perangkat lunak yang menjadi penyusun terbentuknya layanan. Sehingga setiap layanan mampu berjalan dan terhubung seperti pada Gambar 3.4. Tidak hanya menunjukkan hubungan antar layanan, melainkan juga menunjuk-

an susunan layanan pembangun yang saling ketergantungan antara satu layanan dengan layanan lainnya. Penjelasan dari pembuatan layanan dari OpenStack akan dijelaskan pada bagian selanjutnya. Berikut merupakan penjelasan dua bagian penting yang mendasari berjalannya setiap layanan OpenStack :

1. RabbitMQ

Messaging server atau message broker dapat menggunakan RabbitMQ yang bertugas untuk mengkoordinasi informasi dan komunikasi antar server dalam OpenStack.

2. Database

Database digunakan sebagai penyimpanan *metadata* dari setiap layanan OpenStack. Setiap pembuatan layanan dari OpenStack tentunya harus terdaftar terlebih dahulu ke *database*.

### 3.5.1 Keystone (Identity Service)

Keystone atau *identity service* tugas utamanya terbagi dua di antaranya :

1. *User Managent* : Identifikasi *user* dan *permission*.
2. *Service Catalog* : Menyediakan katalog dari *service* yang tersedia dengan menggunakan API (*application programming interface*) *endpoint* dari OpenStack.

Bagian-bagian penting yang harus dimengerti dalam penggunaan keystone seperti dijabarkan pada beberapa point berikut :

1. *User* merupakan representasi dari pengguna layanan, sistem, atau layanan dari OpenStack yang terhubung dengan layanan komputasi awan. Keystone mengatur validasi setiap layanan yang diminta oleh pengguna. Pengguna diharuskan untuk *login*, kemudian memperoleh *token* untuk mengakses layanan tersebut.
2. *Credential* merupakan data pengguna seperti nama pengguna (*username*) dan *password*, nama pengguna dan API *key*, atau sebuah *token authentication* yang disediakan oleh keystone.
3. *Authentication* adalah sebuah konfirmasi untuk mengetahui identitas untuk pengguna yang sedang mengirim permintaan untuk menggunakan dari layanan OpenStack. Data-data kon-

firmasi yang dimasukan oleh pengguna tersebut adalah *credential*.

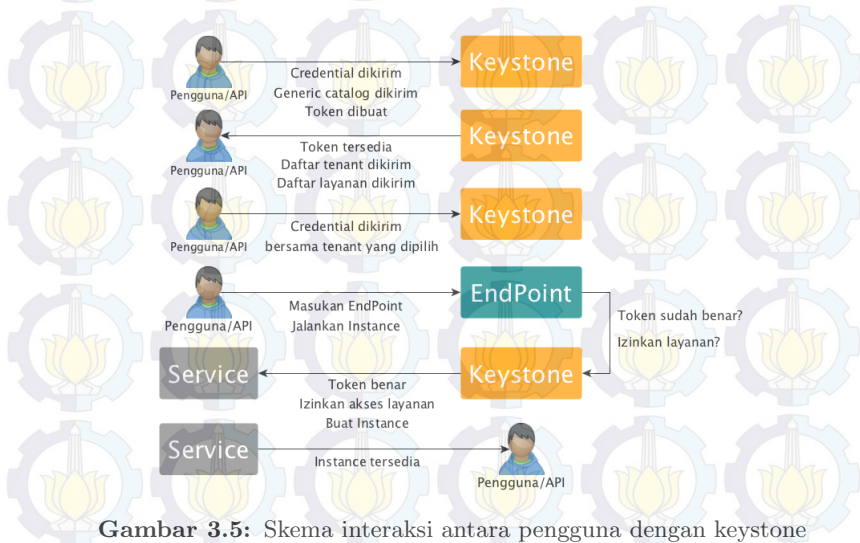
4. *Token* adalah sebuah teks acak yang digunakan untuk mengakses layanan dari OpenStack. Masing-masing *token* mendeskripsikan setiap layanan OpenStack dan *token* akan dihapus dalam jangka waktu tertentu.
5. *Tenant* merupakan sebuah wadah yang digunakan untuk mengelompokkan layanan dari OpenStack yang terhubung dengan keystone. Penelompokan berdasarkan akun, layanan, group atau yang lainnya sesuai dengan kebutuhan.
6. *Service* merupakan istilah yang mewakili layanan dari OpenStack seperti glance, cinder dan nova. Layanan-layanan tersebut mampu menyediakan satu atau lebih *endpoint* yang nantinya bisa diakses oleh pengguna.
7. *Endpoint* adalah sebuah alamat yang dapat diakses oleh pengguna, biasanya berupa URL (*Uniform Resource Locator*).
8. *Role* adalah sebuah aturan yang didefinisikan oleh pengguna untuk melakukan suatu operasi atau perintah tertentu.
9. Keystone *client* adalah sebuah layanan *command line* pada OpenStack. Seperti perintah keystone service-create dan keystone endpoint-create yang digunakan untuk registrasi layanan.

Interaksi antara pengguna dengan keystone yang berperan sebagai pengatur perizinan (otorisasi) dan pengguna (*user*) ditunjukkan pada Gambar 3.5. Di mana pengguna mengirimkan autentikasi ke OpenStack untuk dengan memasukan nama pengguna dan *password*, data inilah yang disebut dengan *credential*. Pengguna yang membuat sebuah *token* secara acak dengan openssl untuk dikirim kepada keystone. Ketika *credential* yang dimasukan oleh pengguna benar, maka layanan dari OpenStack akan berjalan. Sebelum layanan OpenStack berjalan, keystone akan memberikan *list tenant* yang berupa layanan-layanan dari OpenStack untuk dipilih oleh pengguna dan dikirim lagi ke keystone besersamaan dengan *token*.

Kemudian pengguna memilih layanan yang ingin diakses dengan memasukan *endpoint*, bentuk dari *enpoint* adalah berupa URL.



*EndPoint* akan diperiksa kembali tokennya dan layanan apa saja yang ingin diakses oleh pengguna melalui keystone. Setelah itu keystone akan memberi perintah ke layanan tujuan untuk menjalankan *instance* (mesin virtual), akhirnya pengguna dapat mengakses *instance* yang berupa layanan utama dari OpenStack yang berupa mesin virtual tersebut.



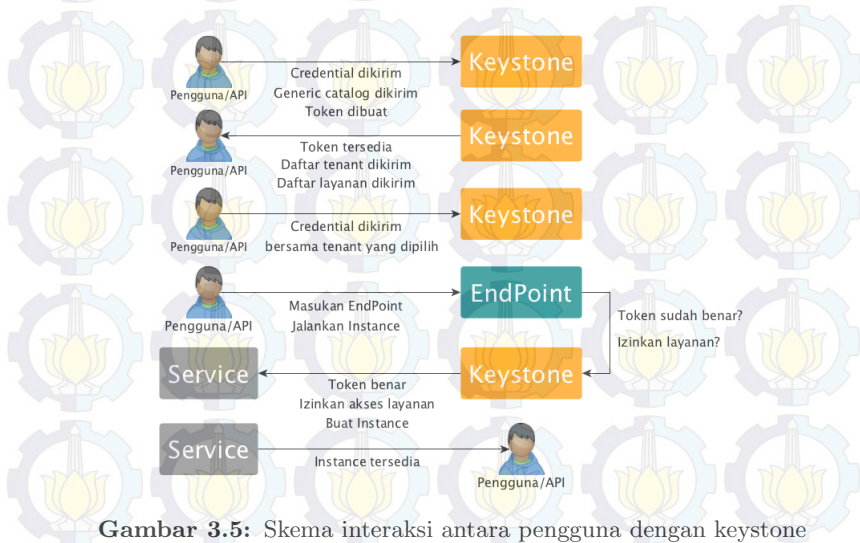
**Gambar 3.5:** Skema interaksi antara pengguna dengan keystone

### 3.5.2 Glance (Image Service)

Glance bertugas untuk pendaftaran, pencarian dan pengambilan *image* dari mesin virtual. Setiap *metadata image* diurutkan atau biasa disebut dengan proses *query* agar memudahkan pencarian *image*. Glance bukan tempat penyimpanan *image*, biasanya disimpan pada *object storage* atau tempat yang lain. Berikut merupakan bagian-bagian penting dari glance :

1. **Glance-api** bertugas menerima perintah simpan, cari dan ambil *image*.
2. **Glance-registry** tugas utamanya adalah melakukan penyimpanan, pemrosesan dan pengambilan metadata seperti tipe dan ukuran *image*.
3. **Database** untuk penyimpanan *metadata* dari *image*.

*EndPoint* akan diperiksa kembali tokennya dan layanan apa saja yang ingin diakses oleh pengguna melalui keystone. Setelah itu keystone akan memberi perintah ke layanan tujuan untuk menjalankan *instance* (mesin virtual), akhirnya pengguna dapat mengakses *instance* yang berupa layanan utama dari OpenStack yang berupa mesin virtual tersebut.



**Gambar 3.5:** Skema interaksi antara pengguna dengan keystone

### 3.5.2 Glance (Image Service)

Glance bertugas untuk pendaftaran, pencarian dan pengambilan *image* dari mesin virtual. Setiap *metadata image* diurutkan atau biasa disebut dengan proses *query* agar memudahkan pencarian *image*. Glance bukan tempat penyimpanan *image*, biasanya disimpan pada *object storage* atau tempat yang lain. Berikut merupakan bagian-bagian penting dari glance :

1. **Glance-api** bertugas menerima perintah simpan, cari dan ambil *image*.
2. **Glance-registry** tugas utamanya adalah melakukan penyimpanan, pemrosesan dan pengambilan metadata seperti tipe dan ukuran *image*.
3. **Database** untuk penyimpanan *metadata* dari *image*.

Pada proses pemasangan *dependence* dari glance terjadi galat di akhir proses, tepatnya saat melakukan perintah `apt-get`. Akibatnya adalah beberapa skrip python pembangun OpenStack gagal tereksekusi, seperti yang ditunjukkan pada Kode 3.1. Pemecahan dari masalah ini adalah penambahan perintah `LC_ALL=en_US.UTF-8` dan `LANG=en_US.UTF-8` pada *environment* sistem operasi.

```
File "/usr/lib/python2.7/dist-packages
/keystone/openstack/common/gettextutils.py",
line 207, in translate_msgid system_locale =
locale.getdefaultlocale()
File "/usr/lib/python2.7/locale.py", line 543, in
getdefaultlocale return_parse_localename (localename)
File "/usr/lib/python2.7/locale.py", line 475,
in_parse_localename raise
ValueError, 'unknown locale: %s' %localename
ValueError: unknown locale: UTF-8
dpkg: error processing package keystone (--configure):
subprocess installed post-installation
script returned error exit status 1
Processing triggers for libc-bin (2.19-0ubuntu6.1)
Processing triggers for ureadahead (0.100.0-16)
Errors were encountered while processing:
glance
E: Sub-process /usr/bin/dpkg returned an error
code (1)
```

### Kode 3.1: Galat saat pemasangan glance

Hal Pertama yang harus dilakukan dalam proses pemasangan glance adalah pembuatan *database*. Kemudian dilanjutkan dengan mendaftarkan glance sebagai layanan pada keystone, tentukan *role*, *tenant* dan *endpoint*. Berkas (*file*) penting yang menghubungkan *database*, keystone dan glance terdapat dalam `/etc/glance/glance-api.conf` seperti yang dituliskan pada Kode 3.2.

```
[DEFAULT]
...
verbose = True

[database]
...
connection = mysql://glance:GLANCE_DBPASS@controller/
glance
```

```
[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = glance
admin_password = GLANCE_PASS

[paste_deploy]
flavor = keystone
```

**Kode 3.2:** Hubungan antara database, keystone dan glance

### 3.5.3 Nova (Compute) pada Controller

Nova atau *compute service* merupakan bagian utama dari IaaS (*Infrastructure as a Service*) sistem yang bertugas sebagai *host* dan sistem manajemen pada komputasi awan. Nova yang terhubung langsung dengan keystone untuk autentikasi, glance untuk *image* dari sistem operasi dan *dashboard* merupakan tempat untuk interaksi antara pengguna dengan layanan dan konfigurasinya. Bagian utama dari nova terdiri dari tiga bagian di antaranya API, *Compute Core*, dan *Networking VMs*. Setiap bagian dari nova terdiri dari beberapa perangkat lunak dasar demi berjalannya nova. Nova yang terpasang pada *controller node* bertugas sebagai pengontrol dari nova yang terpasang pada setiap *compute node*. Berikut merupakan penjelasan setiap bagian perangkat lunak pendukung nova pada *controller node* :

#### API (Application Programming interface)

Pada Gambar 3.4 terlihat API dari nova terhubung dengan glance untuk mengakses dan berkoordinasi perihal *image*. Di sini terjalin hubungan antara horizon (*dashboard*) dengan nova agar layanan OpenStack dapat diakses melalui *browser*. API dari nova terletak pada *controller node*. Berikut merupakan penjelasan perangkat lunak pendukung API pada nova :

1. **Nova-api** berfungsi untuk menerima dan merespon panggilan dari pengguna compute API.
2. **Nova-api-metadata** bertugas menerima permintaan meta-



data dari instances. Nova-api-metadata biasanya digunakan dalam penggunaan compute node dalam jumlah banyak.

## Compute Core

Tugas utama dari *compute core* adalah menyediakan layanan komputasi atau *instance* yang berupa mesin virtual. Jumlah dari *compute core* dapat terus ditambah secara horizontal sesuai kebutuhan komputasi. Berikut ini adalah perangkat lunak pendukung *compute core* :

1. **nova-scheduler** adalah penerima permintaan instance atau mesin virtual dari queue, kemudian menentukan compute node mana yang akan menjalankannya.
2. **nova-conductor** merupakan sebuah perangkat lunak penghubung antara nova-compute dan database.

Sama seperti pada glance, hal pertama yang dilakukan dalam pemasangan nova adalah pemasangan layanan pendukung dan dilanjutkan pembuatan *database*. Kemudian dilanjutkan dengan mendaftarkan nova sebagai layanan pada keystone, tentukan juga *role*, *tenant* dan *endpoint*. File penting yang menghubungkan *database*, keystone, glance dan nova terdapat pada `/etc/nova/nova.conf` seperti yang dituliskan dalam Kode 3.3.

```
[DEFAULT]
...
verbose = True
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
my_ip = 192.168.100.11
vncserver_listen = 192.168.100.11
vncserver_proxyclient_address = 192.168.100.11
network_api_class = nova.network.api.API
security_group_api = nova
compute_driver = libvirt.LibvirtDriver

[database]
...
connection = mysql://nova:NOVADB_PASS@controller/nova

[glance]
...
```



```

host = controller

[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = nova
admin_password = NOVA_PASS

```

**Kode 3.3:** Hubungan database, keystone, galance dan nova

### 3.5.4 Cinder (Block Storage Service)

Cinder bertugas untuk penyediaan penyimpanan tambahan untuk mesin virtual. Pada *block storage service* hanya terdapat satu layanan, yaitu cinder atau *block storage service*. Dalam layanan tersebut terdapat dua layanan pembangun, agar cinder dapat berjalan dan memberikan layanan berupa media penyimpanan. Kedua layanan pembangun tersebut adalah cinder-api, cinder-scheduller dan cinder-volume. Berikut merupakan penjelasan dari ketiga layanan pembangun tersebut :

1. **cinder-api** menerima perintah dan mengarahkannya ke cinder-volume untuk proses eksekusi.
2. **cinder-volume** berinteraksi langsung dengan cinder, dan proses seperti cinder-scheduler. Proses tersebut dilakukan melalui *mesage queue*. Cinder-volume merespon untuk membaca dan menulis permintaan yang kemudian dikirim ke cinder untuk membuat penyimpanan sesuai dengan permintaan pengguna. Cinder-volume dapat berinteraksi dengan berbagai macam media penyimpanan dan berbagai macam driver.
3. **cinder-scheduler** menerima perintah dan mengarahkannya ke cinder-volume untuk proses eksekusi.

Pertama kali yang dilakukan pada proses pemasangan cinder pada *controller node* adalah pemasangan layanan pembangun yang terdiri dari cinder-api, cinder-volume, cinder-scheduler dan dilanjutkan pembuatan *database* untuk cinder. Kemudian dilanjutkan dengan pendaftarkan cinder sebagai layanan pada keystone, tentuk-

an juga *role*, *tenant* dan *endpoint*. *File* penting yang menghubungkan *database*, *keystone* dan *cinder* terdapat pada `/etc/cinder/cinder.conf` seperti yang dituliskan dalam Kode 3.4.

```
[DEFAULT]
...
verbose = True
debug = True
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
auth_strategy = keystone
my_ip = 192.168.100.11
glance_host = controller
volume_driver = cinder.volume.drivers.nfs.NfsDriver
nfs_mount_options = rsize=8192, wsize=8192, timeo=14, intr
nfs_mount_point_base = /var/lib/cinder/mnt
nfs_oversub_ratio = 1.0
nfs_shares_config = /etc/cinder/nfs_share
nfs_sparsed_volumes = False
nfs_used_ratio = 0.95

[database]
...
connection = mysql://cinder:CINDERDB_PASS@controller/
cinder

[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = cinder
admin_password = CINDER_PASS
```

**Kode 3.4:** Hubungan antara *database*, *keystone*, dan *cinder*

Pada Kode 3.4 selain adanya hubungan antara *database*, *keystone* dan *cinder*. Terdapat juga hubungan antara *controller* dengan *storage node*, dimana *controller* terpasang *cinder* sedangkan *storage node* yang terdiri dari *NAS1* dan *NAS2* yang menyediakan layanan *NFS (Network File System)*.

Hubungan antara *storage node* dan *controller node* pada `nfs_shares_config = /etc/cinder/nfs_share`, dimana contro-

lter dapat melakukan *mount* (penambahan penyimpanan) secara otomatis dengan membaca isi *file* yang terletak pada direktori `/etc/cinder/nfs.share`. Isi dari *file* tersebut adalah alamat server dari NFS (`192.168.100.1:/Cinder`). Akses ke NFS server dilakukan melalui alamat `192.168.100.1` milik NAS1. Hal tersebut dikarenakan penggunaan ISCSI yang menggabungkan penyimpanan milik NAS2 ke NAS1 melalui jaringan manajemen.

Pada `volume_driver = cinder.volume.drivers.nfs.NfsDriver` menunjukkan driver penyimpanan yang digunakan oleh cinder, seperti pada pembahasan sebelumnya bahwa jenis penyimpanan yang digunakan adalah NFS. Kemudian pada `nfs_mount_point_base = /var/lib/cinder/mnt` dapat diartikan bahwa pada direktori `/var/lib/cinder/mnt` adalah tempat penyimpanan yang ditambahkan oleh NFS. Selanjutnya pada `nfs_mount_options = rsize=8192, wsize=8192, timeo=14, intr` konfigurasi yang menentukan *permission* dari direktori yang ditambahkan oleh NFS, apakah *read*, *write* dan *execute*.

### 3.6 Pembuatan Compute Node

Pada *compute node* hanya terdapat satu layanan, yaitu nova atau *compute service*. Dalam layanan tersebut terdapat dua layanan pembangun, agar nova dapat berjalan dan layanan mesin virtual (*instance*) dapat digunakan. Kedua perangkat lunak itu adalah nova-compute dan nova-network. Nova-compute berada pada bagian *compute core* dan nova-network dan nova-dhcpbridge pada *networking VMs*, berikut penjelasannya :

1. **nova-compute** bertugas menjalankan proses pembuatan dan penghapusan layanan yang berupa mesin virtual melalui hypervisor API seperti pada Gambar 3.4. Contoh dari hypervisor API adalah XenAPI untuk XenServer/XCP, libvirt untuk KVM atau QEMU, VMware API untuk VMware. Dalam penelitian ini hypervisor API yang dipakai adalah libvirt untuk KVM atau QEMU. Letak dari nova-compute berada pada *compute node*, jumlahnya terus bisa ditambah sesuai kebutuhan komputasi.
2. **nova-network** berkerja mirip seperti nova-compute, dengan menerima permintaan layanan berupa kebutuhan jaringan dan

menjalankannya untuk memenuhi permintaan tersebut. Seperti konfigurasi *bridge* dan perubahan aturan pada *iptables* untuk mengatur setiap *instance*.

3. **nova-dhcpbridge** merupakan sebuah skrip yang berfungsi untuk mendeteksi IP Address yang disimpan dalam database dengan menggunakan *dnsmasq* yang merupakan fitur dari *dhcp-script*.

Pertama kali yang dilakukan dalam pemasangan nova pada *compute node* adalah pemasangan layanan pembangun yaitu *nova-compute*. Selanjutnya tambahkan konfigurasi pada *file* yang menghubungkan seluruh *compute node* dengan *controller node*. File tersebut terdapat pada `/etc/nova/nova.conf` seperti yang tertulis dalam Kode 3.5.

```
[DEFAULT]
...
my_ip = 192.168.100.21
vnc_enabled = True
vncserver_listen = 0.0.0.0
vncserver_proxyclient_address = 192.168.100.21
novncproxy_base_url =
http://controller:6080/vnc_auto.html
verbose = True

[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = nova
admin_password = NOVA_PASS

[libvirt]
...
virt_type = qemu

[glance]
...
host = controller
```

**Kode 3.5:** Hubungan nova pada *compute node* dengan *controller node*



Pembuatan nova-network pada *compute node* diawali dengan pemasangan *dependence*. Kemudian tambahkan konfigurasi pada *file* yang menghubungkan antara nova pada *controller node* dengan nova pada *compute node*. Hal tersebut bertujuan agar *controller node* dapat terhubung dan memberikan perintah ke semua *compute node*. Kedua *file* tersebut terdapat pada `/etc/nova/nova.conf`. Kode 3.6 merupakan konfigurasi yang harus ditambahkan pada *controller node* dan Kode 3.7 adalah konfigurasi yang harus ditambahkan pada setiap *compute node*.

```
[DEFAULT]
...
network_api_class = nova.network.api.API
security_group_api = nova
```

**Kode 3.6:** Nova-network pada *controller node*

```
[DEFAULT]
...
network_api_class = nova.network.api.API
security_group_api = nova
firewall_driver = nova.virt.libvirt.firewall.
    IptablesFirewallDriver
network_manager = nova.network.manager.FlatDHCPManager
network_size = 254
allow_same_net_traffic = False
multi_host = True
send_arp_for_ha = True
share_dhcp_address = True
force_dhcp_release = True
flat_network_bridge = br100
flat_interface = em2
public_interface = em2
```

**Kode 3.7:** Nova-network pada *compute node*

### 3.7 Pembuatan Horizon (Dashboard)

Openstack *dashboard* disebut juga dengan Horizon adalah sebuah antarmuka berbasis *website* yang berfungsi untuk memudahkan pengguna dalam mengakses setiap layanan dari Openstack. Kemudian mempermudah pengguna dalam mengatur sumber daya dari

untuk komputasi yang digunakannya, seperti memori, penyimpanan dan komputasi (CPU). Selain itu horizon juga mengatur interaksi antara *compute* dengan *controller* melalui Openstack API agar dapat diakses dan diatur melalui *browser*. Pada penggunaan dan pengaksesannya, pengguna diharuskan memakai *browser* yang mendukung HTML5, mengizinkan *cookies* dan javascript.

Beberapa aplikasi pendukung untuk membangun horizon diantaranya apache2 untuk *web server*, memcache dan python versi 2.6 atau 2.7 yang mendukung Django. Setelah pemasangan aplikasi pendukung tersebut dilakukan, tambahkan konfigurasi pada *file* yang dapat menjadikan *controller node* dapat diakses melalui antarmuka web. File tersebut terdapat pada `/etc/openstack-dashboard/local_settings.py` seperti pada Kode 3.8.

```
...
OPENSTACK_HOST = "controller"
...
ALLOWED_HOSTS = ['*']
...
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.
MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    }
}
```

**Kode 3.8:** Penghubung *controller node* dengan antarmuka *web*

## BAB 4

# PENGUJIAN DAN ANALISA

Pada bab ini pengujian dibagi menjadi lima tahapan yaitu pengujian layanan (*service*), pengujian jaringan, kemampuan maksimal komputasi awan, perbandingan performa mesin virtual dan perhitungan jumlah *thread*. Sehingga dengan adanya pengujian tersebut, dapat ditarik beberapa kesimpulan dari pelaksanaan tugas akhir ini.

### 4.1 Pengujian Layanan

Pada bagian ini pengujian dilakukan untuk memastikan apakah layanan pada komputasi awan dapat berjalan atau tidak. Layanan yang akan diuji dalam penelitian ini diantaranya keystone, glance, nova dan cinder yang akan dijabarkan pada bagian selanjutnya. Kemudian dilanjutkan dengan pengujian pemasangan sistem operasi pada mesin virtual (*instance*) yang disediakan oleh komputasi awan.

Penggunaan layanan akan dibandingkan keberhasilannya melalui *console* dengan melakukan *remote* secara langsung ke *controller node*, dan penggunaan layanan yang diakses melalui antarmuka berbasis *website*. Antarmuka tersebut merupakan salah satu layanan utama dari OpenStack yang biasanya disebut dengan horizon (*dashboard*).

#### 4.1.1 Uji Coba Keystone (Identity Service)

Terdapat dua pengguna yang dibuat pada proses implementasi yaitu *admin* dan *user*. Selain itu setiap layanan juga didaftarkan sebagai pengguna dengan *tenant* dan *role* yang berbeda. Pengujian pertama kali yang harus dilakukan adalah masuk (*login*) ke dalam sistem manajemen OpenStack, *login* adalah bagian dari layanan yang disediakan oleh keystone.

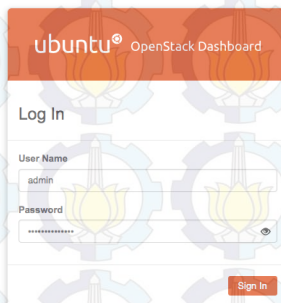
Pengguna terlebih dahulu diharuskan melakukan *remote* ke *controller node*. Kemudian membuat bash script seperti pada Kode 4.1 dan menerapkannya ke *environment* sistem operasi untuk *login*. Penggunaan perintah `source admin-openstack.sh` untuk admin

atau `source (nama script).sh` untuk pengguna lain dengan mengisi `OS_TENANT_NAME`, `OS_USERNAME`, dan `OS_PASSWORD` yang berbeda sesuai dengan nama dan *password* dari pengguna.

```
export OS_TENANT_NAME = admin
export OS_USERNAME = admin
export OS_PASSWORD = ADMIN_LOGIN_PASS
export OS_AUTH_URL = http://controller:35357/v2.0
```

**Kode 4.1:** Login script pada keystone dengan nama pengguna admin.

Pengujian keystone melalui antarmuka (*user interface*) ditunjukkan pada Gambar 4.1 dimana pengguna dapat memasukkan nama pengguna (*username*) dan *password* yang telah dibuat sebelumnya. Pengaksesan halaman tersebut dilakukan melalui *browser*, pengguna harus terlebih dahulu mengakses alamat `http://192.168.100.11/horizon`, di mana halaman tersebut merupakan alamat dari *controller node*. Proses pembuatan *username* dan *password* akan dibahas juga pada bagian ini juga.



**Gambar 4.1:** *Login screen*

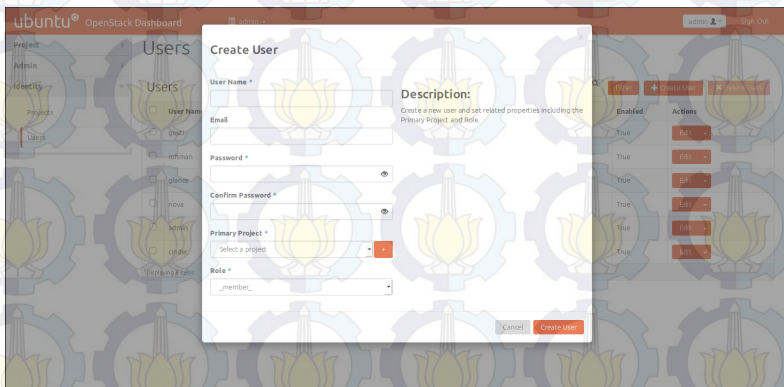
Penggunaan perintah keystone yang lebih spesifik sesuai dengan kebutuhan. Pada Kode 4.2 merupakan perintah untuk penambahan pengguna atau layanan. Sebagai contoh, nama pengguna yang ingin ditambahkan dalam pengujian ini adalah “rohman”. Maka pengguna diharuskan untuk menyesuaikan nama, *password* dan *email* pada skrip tersebut.



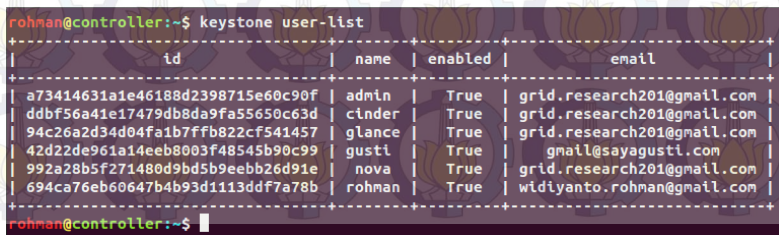
```
$ keystone user-create --name rohman --pass ROHMAN_PASS \
--email ROHMAN_EMAIL_ADDRESS
```

#### Kode 4.2: Perintah untuk tambah pengguna

Penambahan pengguna atau layanan melalui antarmuka, *admin* diharuskan memilih menu **Identity** kemudian dilanjutkan dengan **User** dan pilih **Create User** seperti pada Gambar 4.2. Setelah melakukan penambahan pengguna melalui antarmuka, nama pengguna akan tampil pada halaman itu juga. Pada Gambar 4.3 merupakan perintah untuk melihat tabel daftar pengguna, pada gambar tersebut juga dapat dilihat pengguna yang berhasil ditambahkan.



Gambar 4.2: Penambahan pengguna melalui antarmuka web



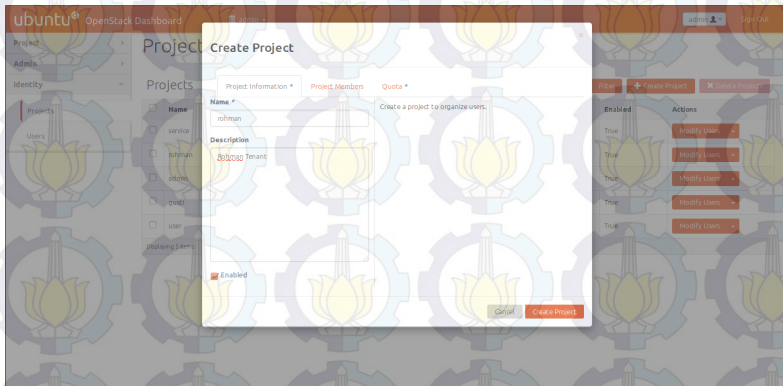
Gambar 4.3: Daftar pengguna dan layanan melalui *console*

Penambahan *tenant* atau *project* dapat dilakukan dengan perintah seperti pada Kode 4.3, dimana dalam perintah tersebut terdapat nama *tenant* dan deskripsinya. Sebagai contoh, nama *tenant* yang ingin ditambahkan dalam pengujian ini adalah “rohman” dengan deskripsi “Rohman Tenant”.

```
$ keystone tenant-create --name rohman --description \
"Rohman Tenant"
```

**Kode 4.3:** Perintah untuk tambah *tenant* atau *project*

Ketika penambahan *tenant* dilakukan melalui antarmuka, yang harus dilakukan oleh *admin* adalah memilih menu **Identity**, kemudian **Project** dan pilih **Create Project**. Kemudian pada **Project Information** isikan nama *tenant* dan deskripsinya seperti pada Gambar 4.4.



**Gambar 4.4:** Daftar pengguna dan layanan melalui antarmuka

Setelah melakukan penambahan *tenant* melalui antarmuka, nama *tenant* dan deskripsinya akan tampil pada tabel daftar di halaman tersebut. Sedangkan pada Gambar 4.5 merupakan perintah untuk melihat daftar *tenant* melalui *console*, pada gambar tersebut juga terlihat *tenant* yang berhasil dibuat. Selanjutnya adalah penambahan *role* ke *tenant* dan pengguna yang dibuat sebelumnya, *role* disini adalah peran dari pengguna. Apakah pengguna tersebut

berperan sebagai *admin* atau *member*. Pada Kode 4.4 merupakan langkah penambahan *role* dengan menggunakan perintah melalui *console*.

```
rohman@controller:~$ keystone tenant-list
```

id	name	enabled
217bf01ccd7747368e022cb9b6a34a75	admin	True
ab86185113d649eb8119bcb4e7db0f5c	gusti	True
18f3ebf7da75485db204b8167efd600f	rohman	True
048ee86912d54a848c461791a6147d0d	service	True
d0159b5f3cd4127870d7180696040eb	user	True

```
rohman@controller:~$
```

Gambar 4.5: Daftar *tenant* melalui *console*

```
$ keystone user-role-add --tenant rohman --user rohman \
--role _member_
```

**Kode 4.4:** Perintah untuk memberi *role* ke sebuah *project*

Pada penambahan *role* ke *tenant* dan pengguna dilakukan melalui antarmuka, yang harus dilakukan oleh *admin* adalah memilih menu **Identity**, kemudian **Project** dan selanjutnya pilih **Create Project**. Pada menu **Create Project** pilih *tab* **Project Member** seperti pada Gambar 4.6. Kemudian pilih pengguna yang akan digunakan. Pada menu *dropdown* pilih *\_member\_* untuk menjadikan pengguna tersebut sebagai *member* atau *admin* untuk mejadikan-nya sebagai *admin*.



Gambar 4.6: Pengguna dan layanan saat pembuatan *project*

Setelah melakukan penambahan *role* ke *tenant* dan pengguna melalui antarmuka, nama *tenant* dan deskripsinya akan tampil pada halaman itu juga. Pada Gambar 4.7 merupakan perintah untuk melihat daftar *role*, yang digunakan untuk mengatur pengaturan hak akses (*permission*) dari pengguna.

```
rohman@controller:~$ keystone role-list
```

id	name
e63bf78394d04225a82fe2a0a4a4170a	_member_
1bb942fddca2423492c81e49e4332590	admin

```
rohman@controller:~$
```

Gambar 4.7: Daftar pengguna dan *role* melalui *console*

Mengacu pada proses yang dijelaskan pada bagian ini, maka dapat disimpulkan bahwa pemasangan keystone sukses. Keystone sudah mampu menyediakan layanan *authentication* mulai dari, proses *login*, pembuatan *user*, *tenant* dan penambahan *role*.

#### 4.1.2 Uji Coba Glance (Image Service)

Glance merupakan layanan penyedia *image* sistem operasi. Pada pengujian ini yang akan dilakukan adalah pengunggahan (*upload*) *file image* sistem operasi ke glance. Pada pengujian ini sistem operasi yang akan diunggah adalah ubuntu-trusty-tahr-14.04-amd64. Seperti pengujian pada keystone dibagian sebelumnya, hal pertama yang harus dilakukan saat melakukan pengujian melalui *console* adalah masuk menggunakan bash skrip yang terlebih dahulu harus dibuat seperti pada Kode 4.1. Selanjutnya gunakan perintah untuk upload image seperti pada Kode 4.5

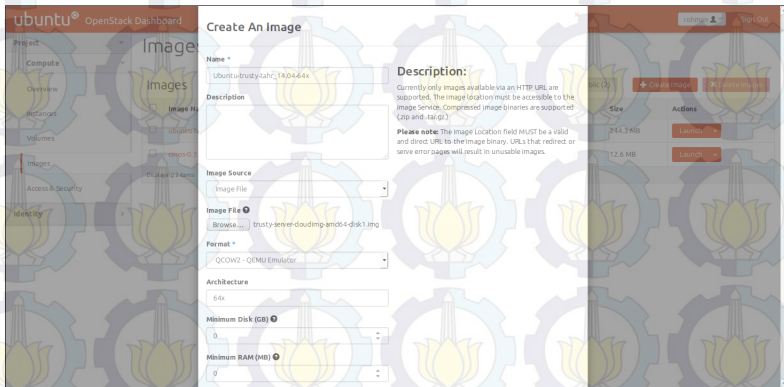
```
$ glance image-create \  
--name "ubuntu-trusty-tahr-14.04-amd64" \  
--file ubuntu-trusty-tahr-14.04-amd64 \  
--disk-format qcow2 --container-format bare \  
--is-public True --progress
```

Kode 4.5: Perintah untuk mengunggah *image* ke glance

Pada Kode 4.5 dapat ditarik kesimpulan bahwa sistem operasi yang diunggah adalah ubuntu versi 14.04 (Trusty Tahr) dengan ti-



pe *disk* *qcow2* yang biasanya digunakan oleh mesin virtual QEMU. Agar dapat diakses oleh seluruh pengguna maka menu *public* diatur sebagai *True*. Untuk melihat apakah *image* sudah berhasil terunggah, gunakan perintah pada Gambar 4.9. Selain itu akan tampil daftar *image* yang pernah diunggah.



Gambar 4.8: Penambahan *image* melalui antarmuka

Pada pengujian glance menggunakan antarmuka, yang harus dilakukan oleh pengguna setelah masuk adalah memilih menu **Project**, kemudian **Compute** dan pilih menu **Image**. Lanjutkan dengan **Create Image** kemudian isi *form* sesuai spesifikasi image, dan unggah *file image* seperti pada Gambar 4.8, kemudian pilih **Create Image**. Setelah berhasil melakukan pengunggahan *Image* melalui antarmuka, nama *Image* dan deskripsinya akan tampil pada tabel *image* halaman itu juga.

Berdasarkan pada setiap langkah pada bagian ini, dimana pengguna berhasil melakukan pengunggahan *image* ke dalam sistem komputasi awan melalui glance. Proses pengunggahan image yang dilakukan melalui console dan antarmuka, keduanya berhasil dilakukan. Maka dapat ditarik kesimpulan bahwa pemasangan glance sukses.

```

rohan@controller:~$ glance image-list
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | | Name | | Disk Format | | Container Format | | Size | | Status |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 017c66c1-2011-41d2-978a-b6f94e225609 | | cirros-0.3.3-x86_64 | | qcow2 | | bare | | 13200896 | | active |
| 207a62b4-46f5-4103-8f58-032da8bc5231 | | ubuntu-trusty-tahr-14.04-and64 | | qcow2 | | bare | | 256180736 | | active |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
rohan@controller:~$ █

```

Gambar 4.9: Daftar *image* yang berhasil terunggah dilihat melalui console

```

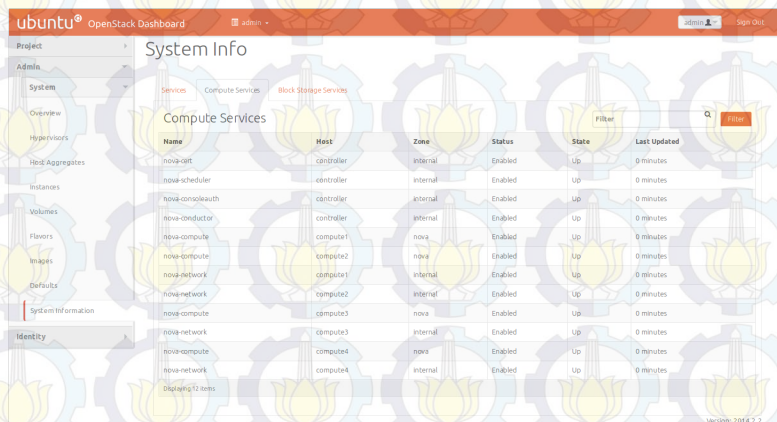
rohan@controller:~$ nova service-list
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Id | Binary | Host | Zone | Status | State | Updated_at | | Disabled Reason |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | nova-cert | controller | Internal | enabled | up | 2015-05-06T06:32:28.000000 | | - |
| 2 | nova-scheduler | controller | Internal | enabled | up | 2015-05-06T06:32:28.000000 | | - |
| 3 | nova-consoleauth | controller | Internal | enabled | up | 2015-05-06T06:32:27.000000 | | - |
| 4 | nova-conductor | controller | Internal | enabled | up | 2015-05-06T06:32:24.000000 | | - |
| 5 | nova-compute | compute1 | nova | enabled | up | 2015-05-06T06:32:27.000000 | | None |
| 6 | nova-compute | compute2 | nova | enabled | up | 2015-05-06T06:32:24.000000 | | None |
| 7 | nova-network | compute1 | Internal | enabled | up | 2015-05-06T06:32:29.000000 | | - |
| 8 | nova-network | compute2 | Internal | enabled | up | 2015-05-06T06:32:24.000000 | | - |
| 9 | nova-compute | compute3 | nova | enabled | up | 2015-05-06T06:32:27.000000 | | None |
| 10 | nova-network | compute3 | Internal | enabled | up | 2015-05-06T06:32:27.000000 | | - |
| 11 | nova-network | compute4 | Internal | enabled | up | 2015-05-06T06:32:24.000000 | | - |
| 12 | nova-network | compute4 | Internal | enabled | up | 2015-05-06T06:32:24.000000 | | - |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
rohan@controller:~$ █

```

Gambar 4.10: Daftar layanan dari nova yang sedang berjalan dilihat melalui console

### 4.1.3 Uji Coba Nova (Compute Service)

Pengujian pada Nova atau *compute service* dilakukan dengan melakukan pengecekan apakah seluruh layanan yang mendasari nova sudah terpasang dan tidak menunjukkan status galat. Pengecekan dilakukan dengan menggunakan perintah pada *console* seperti pada Gambar 4.10. Sedangkan pengecekan melalui antarmuka dapat dilakukan dengan memilih menu **Admin**, kemudian **System**, dan dilanjutkan dengan memilih **System Information**. Pada *tab Compute Services* akan terlihat seluruh status dari layanan yang membangun nova seperti pada Gambar 4.11. Selain itu terlihat juga nova sukses tersambung dengan glance seperti pada Gambar 4.12, yang mana *image* dari glance terdeteksi oleh nova.



Name	Host	Zone	Status	State	Last Updated
nova-api	controller	internal	Enabled	Up	0 minutes
nova-scheduler	controller	internal	Enabled	Up	0 minutes
nova-conductor	controller	internal	Enabled	Up	0 minutes
nova-compute	compute1	nova	Enabled	Up	0 minutes
nova-compute	compute2	nova	Enabled	Up	0 minutes
nova-network	compute1	internal	Enabled	Up	0 minutes
nova-network	compute2	internal	Enabled	Up	0 minutes
nova-compute	compute3	nova	Enabled	Up	0 minutes
nova-network	compute3	internal	Enabled	Up	0 minutes
nova-compute	compute4	nova	Enabled	Up	0 minutes
nova-network	compute4	internal	Enabled	Up	0 minutes

**Gambar 4.11:** Daftar layanan dari nova yang sedang berjalan, bila dilihat melalui antarmuka *web*



```
rohman@controller:~$ nova image-list
```

ID	Name	Status	Server
017c66c1-2011-41d2-978a-b6f94e225609	clirros-0.3.3-x86_64	ACTIVE	
207a62b4-46f5-4103-8f58-032da8bc5231	ubuntu-trusty-tahr-14.04-amd64	ACTIVE	

```
rohman@controller:~$
```

**Gambar 4.12:** Daftar *image* sistem operasi dari glance yang terbaca oleh nova

#### 4.1.4 Uji Coba Mesin Virtual (Instance)

Mesin virtual merupakan layanan utama dari komputasi awan yang dibangun untuk penelitian ini. Pada bagian ini dilakukan pengujian, apakah layanan dari komputasi awan yang berupa mesin virtual bisa digunakan. Langkah awal yang dilakukan adalah penerapan *authentication*, pengguna, *tenant*, dan *endpoint* yang merupakan layanan utama dari keystone seperti pada bagian sebelumnya. Kemudian dilanjutkan dengan glance yang sudah terhubung dengan nova. Uji coba instance dilakukan menggunakan dua cara, yaitu melalui *console* dan antarmuka.

Masuk dengan menuliskan nama pengguna dan password, jika menggunakan antarmuka. Jika menggunakan *console*, maka buat bash skrip untuk masuk seperti pada Kode 4.1. Kemudian tambahkan key pair pada “rohman” untuk mengakses mesin virtual. Buat sebuah *key pair* dengan perintah *ssh-keygen*, selanjutnya tambahkan ke nova dengan perintah seperti Kode 4.6. Gunakan perintah seperti pada Gambar 4.13 untuk melihat apakah *keypair* sudah berhasil ditambahkan.

```
$ nova keypair-add --pub-key ~/.ssh/id_rsa.pub rohman-key
```

**Kode 4.6:** Perintah untuk menambahkan *keypair* ke nova

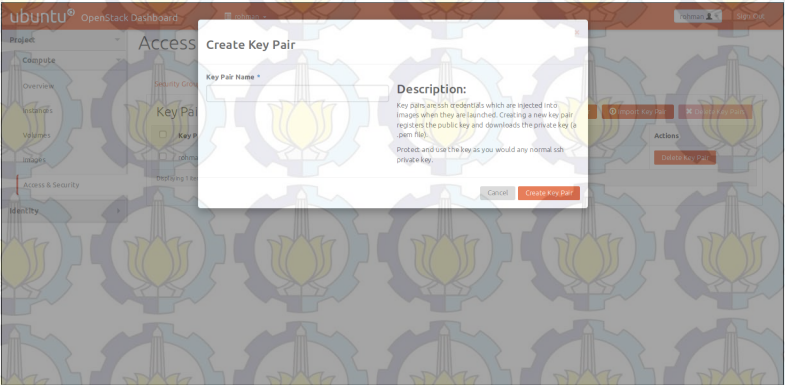
```
rohman@controller:~$ nova keypair-list
+-----+-----+
| Name   | Fingerprint |
+-----+-----+
| rohman-key | eb:82:f1:64:50:78:be:02:80:aa:db:cd:2e:f5:4b:d7 |
+-----+-----+
```

**Gambar 4.13:** Daftar *key pairs* melalui *console*

Penambahan *keypair* melalui antarmuka. Pengguna diharuskan memilih menu **Project**, kemudian **Compute** dan dilanjutkan dengan memilih menu **Access & Security**. Pada *tab* **Key Pairs** pilih **Create Key Pairs** dan masukan nama *key pair* seperti pada Gambar 4.14. Setelah proses penambahan selesai, maka akan muncul *key pairs* baru pada tabel *key pair* pada halaman tersebut, sebelum itu akan muncul sebuah halaman untuk mengunduh *key*



*pair* pada browser yang digunakan oleh pengguna. *Key pair* sangat penting, peran dari *key pair* sendiri seperti kunci yang digunakan untuk melakukan *remote* atau SSH (*Secure Shell*) ke mesin virtual yang akan dijelaskan pada bagian ini selanjutnya.



Gambar 4.14: Pembuatan *keypair* melalui antarmuka

Selanjutnya pilih *flavor* yang disediakan oleh nova, *flavor* adalah pilihan paket spesifikasi perangkat keras yang akan digunakan oleh mesin virtual seperti yang terlihat pada Gambar 4.15. Pada pengujian ini paket yang dipilih adalah **m1.small**, dengan ukuran 1 GB untuk memori (RAM) dan 20 GB untuk penyimpanan. Melihat daftar *flavor* melalui antarmuka, pengguna dapat mengaksesnya pada saat menjalankan mesin virtual (*launch instance*) seperti pada Gambar 4.19.

```
rohman@controller:~$ nova flavor-list
```

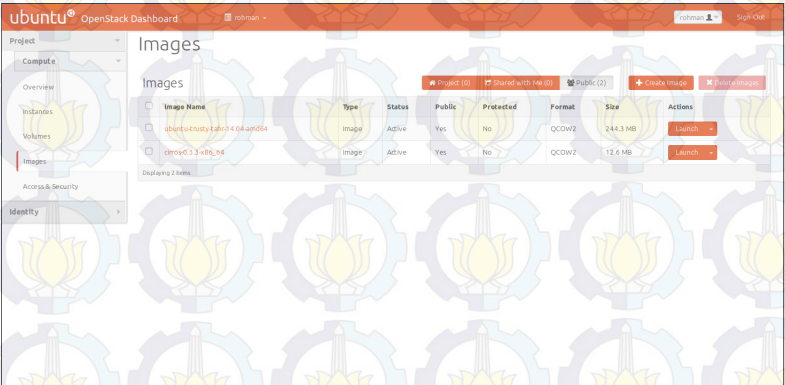
ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor	Is_Public
1	m1.tiny	512	1	0		1	1.0	True
2	m1.small	2048	20	0		1	1.0	True
3	m1.medium	4096	40	0		2	1.0	True
4	m1.large	8192	80	0		4	1.0	True
5	m1.test	1024	2199	0		255	1.0	True
6	m1.disk	2048	40	0		8	1.0	True

```
rohman@controller:~$
```

Gambar 4.15: Daftar *flavor* melalui *console*

Pemilihan sistem operasi untuk mesin virtual ditunjukkan oleh Gambar 4.12 pada bagian uji coba nova. Di mana pengguna dapat

memilih satu sistem operasi untuk dipasang dari pilihan yang disediakan. Sedangkan untuk melihat daftar image melalui antarmuka, pengguna cukup memilih **Project**, kemudian **Compute** dan pilih **Image** seperti pada Gambar 4.16.



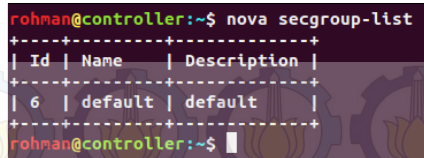
Gambar 4.16: Daftar *image* melalui antarmuka web

Nova juga menyediakan layanan jaringan bagi *instance*, setiap *instance* akan memperoleh alamat IP secara otomatis atau DHCP (*Dynamic Host Configuration Protocol*) melalui nova-network. Pada Gambar 4.17 menunjukan tentang alamat IP yang dialokasikan untuk seluruh pengguna komputasi awan dengan perintah `nova net-list`, tetapi perintah tersebut hanya bisa dijalankan oleh *admin* dengan script `admin-opensh.sh`.

```
rohman@controller:~$ nova net-list
+-----+-----+-----+
| ID                | Label | CIDR                |
+-----+-----+-----+
| 3e149db9-5f3e-4dfb-ac7b-23e07c4e2dfe | nova-net | 192.168.100.128/25 |
+-----+-----+-----+
rohman@controller:~$
```

Gambar 4.17: Daftar *nova-network* melalui *console*

Terdapat juga *secgroup* (*security group*) untuk menjaga keamanan *instance*. Pada penelitian ini *secgroup* yang digunakan adalah *secgroup* standar dari nova yang bekerja mirip dengan *firewall*. Gunakan perintah `nova secgroup-list` untuk melihat *secgroup* default seperti pada Gambar 4.18.



```

rohman@controller:~$ nova secgroup-list
+-----+-----+-----+
| Id | Name | Description |
+-----+-----+-----+
| 6 | default | default |
+-----+-----+-----+
rohman@controller:~$

```

**Gambar 4.18:** Daftar secgroup melalui *console*

Mengacu pada langkah-langkah sebelumnya untuk menjalankan *instance*, dengan masukan paket yang dipilih kemudian pilih sistem operasi yang akan digunakan, nomor ID (*Identification*) dari nova-network, security-group, security-key dan nama dari mesin virtual yang akan dibuat seperti pada Kode 4.7. Setelah perintah ini dijalankan maka akan tampil informasi dari mesin virtual yang telah sukses dibuat. Gunakan perintah `nova list` pada *console* untuk melihat status dari mesin virtual seperti pada Gambar 4.26. Saat mesin virtual galat atau berhasil dipasang juga dijelaskan pada tabel tersebut.

```

$ nova boot --flavor m1.small \
--image ubuntu-trusty-tahr-14.04-amd64 \
--nic net-id=3e149db9-5f3e-4dfb-ac7b-23e07c4e2dfe \
--security-group default --key-name rohman-key Ubuntu1

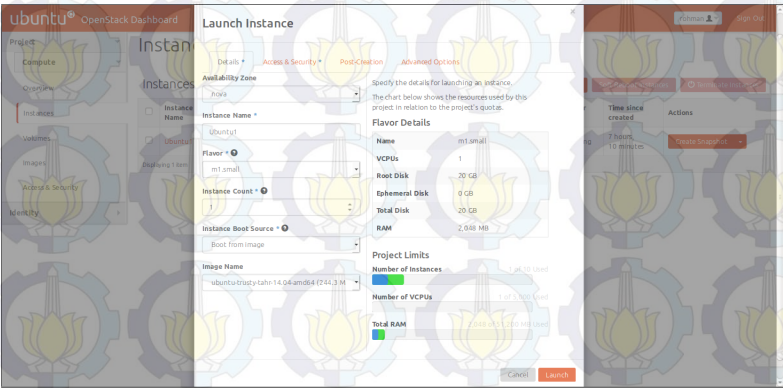
```

**Kode 4.7:** Perintah untuk membuat mesin virtual (*instance*) melalui *console*

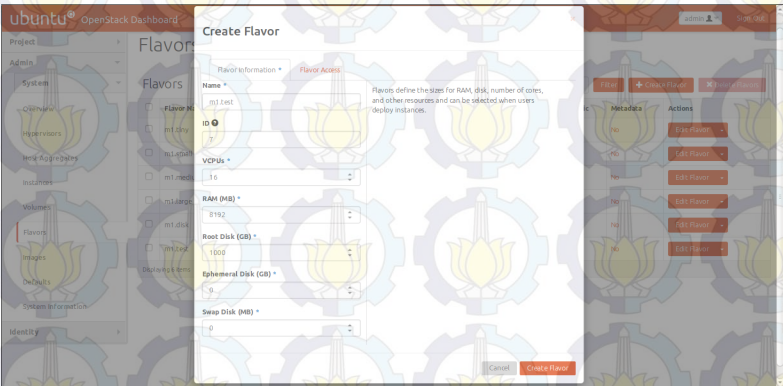
Menjalankan mesin virtual dengan antarmuka seperti pada Gambar 4.19 yang harus dilakukan adalah pilih **Project**, kemudian **Compute** dan dalam menu **Instance** pilih **Launch Instance**. Pada menu yang berupa *form* dan *dropdown*, isi dan sesuaikan dengan spesifikasi yang dibutuhkan. Contohnya nama “Ubuntu1”, selanjutnya pilih *flavor* sesuai dengan spesifikasi yang dibutuhkan misalkan **m1.small**. Kemudian masukan jumlah mesin virtual yang ingin dibuat pada *form* **Instance Count**. Pada **Instance Boot Source** pilih media yang digunakan untuk memasang sistem operasi. Misalkan **From Image**, maka nova akan terhubung dengan glance untuk membaca *image* dari sistem operasi yang disediakan.

Pengaturan spesifikasi untuk pembuatan mesin virtual dapat dilakukan dengan menambahkan *flavor* sesuai dengan kebutuhan

pengguna. Tetapi dalam penambahan *flavor* hanya bisa dilakukan oleh admin dengan mengisi *form* dan menu *dropdown* seperti pada Gambar 4.20. Setelah masuk sebagai admin, kemudian pilih menu **Admin**. Kemudian lanjutkan dengan memilih **Flavor** dan **Create Flavor**. Setelah *flavor* ditambahkan, maka *flavor* baru akan tampil pada halaman itu juga.



Gambar 4.19: Pembuatan mesin virtual melalui antarmuka

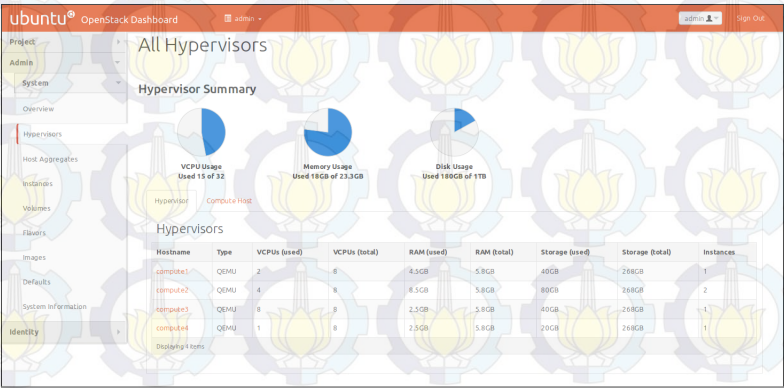


Gambar 4.20: Penambahan *flavor* melalui antarmuka

OpenStack memiliki berapa sistem monitoring untuk meman-  
tau kondisi layanan dan sumber daya pada menu **Admin**, con-



tohnya adalah **Overview**, **Hypervisors**, **Host Aggregates**, dan **System Information**. Memori, penyimpanan, dan CPU adalah contoh sumber daya yang dimonitorng secara keseluruhan agar *server* tetap bisa berjalan dengan baik. Pada Gambar 4.21 merupakan tampilan dari monitoring pada menu **hypervisors**, yang bertugas memonitor kondisi sumber daya pada setiap komputer server.



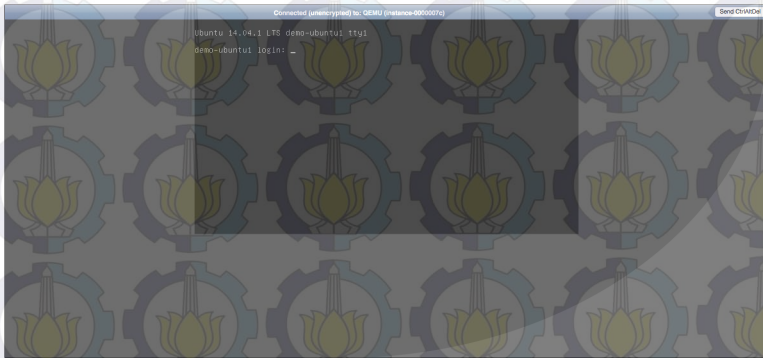
Gambar 4.21: Tampilan monitoring *hypervisors*

```
rohman@controller:~$ nova get-vnc-console Ubuntu1 novnc
+-----+
| Type | Url |
+-----+
| novnc | http://controller:6080/vnc_auto.html?token=b948a325-08e3-437a-865c-7ea6aae178fc |
+-----+
rohman@controller:~$
```

Gambar 4.22: Perintah tampilan alamat VNC yang berupa URL

Pengaksesan mesin virtual dapat dilakukan melalui banyak cara diantaranya menggunakan VNC (*Virtual Network Computing*). VNC memungkinkan pengguna dapat melakukan akses ke mesin virtual melalui *browser*. Pengujian pengaksesan mesin virtual dengan VNC dilakukan menggunakan dua cara, yaitu melalui *console* dan antarmuka web. Gunakan perintah `nova get-vnc-console Ubuntu1 novnc`, Ubuntu1 merupakan nama mesin virtual yang akan diakses seperti pada Gambar 4.22. Tampilah pada *console* alamat VNC dari mesin virtual yang berupa URL dan dapat diakses melalui *browser* seperti pada Gambar 4.23.

Penggunaan antarmuka *web* untuk mengakses VNC, pengguna dapat memilih menu **Project**. Kemudian pada menu **Compute** pilih **Instance**, pilih nama mesin virtual yang ingin diakses. Selanjutnya di dalam *tab console* pilih **Click here to show console only**, maka akan tampil sebuah *console* yang dapat dipakai melalui *browser*. Pada setiap pengaksesan alamat VNC yang berupa URL pada *browser*, terlebih dahulu rubah hostname *controller* dengan alamat IP dari *controller node*. Hal ini juga berlaku pada proses pengaksesan melalui *console* untuk memperoleh alamat VNC yang berupa URL.



**Gambar 4.23:** Akses mesin virtual melalui *browser*

Berbeda dengan pengaksesan mesin virtual dengan VNC yang dapat diakses melalui *browser*, dimana pengguna dapat memberikan perintah atau kontrol terhadap sistem operasi melalui antarmuka. Pengaksesan mesin virtual melalui *remote* atau SSH, pengguna harus mengatur konfigurasi dari secgroup. Di mana secgroup bekerja mirip seperti *firewall*, pengguna harus memberi izin agar mesin virtual dapat diakses dari luar jaringan sistem komputasi awan menggunakan *remote* atau SSH.

Pada Kode 4.8 merupakan perintah untuk memberi izin agar mesin virtual dapat diakses melalui SSH. Bukan hanya pengaturan untuk pengaksesan melalui SSH, banyak protokol TCP/IP dan port lainnya dapat diatur. Misalnya ICMP (*Internet Control Message Protocol*) atau yang biasa disebut dengan PING, pada Ko-

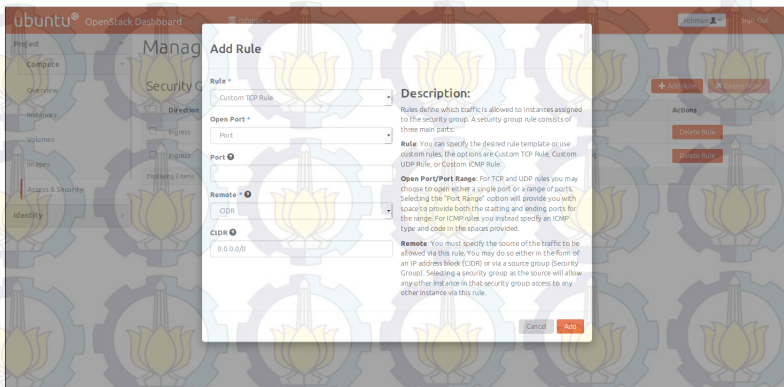
de 4.9 merupakan perintah untuk memberi izin agar mesin virtual dapat menerima paket ICMP dari luar. Pada perintah tersebut perintah yang digunakan adalah `nova secgroup-add-rule`, kemudian diikuti dengan nama *security group*, protokol dan *subnet* jaringan.

```
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

**Kode 4.8:** Perintah untuk memberikan izin agar mesin virtual dapat diakses melalui SSH

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
```

**Kode 4.9:** Perintah untuk memberikan izin agar mesin virtual dapat menerima paket ICMP

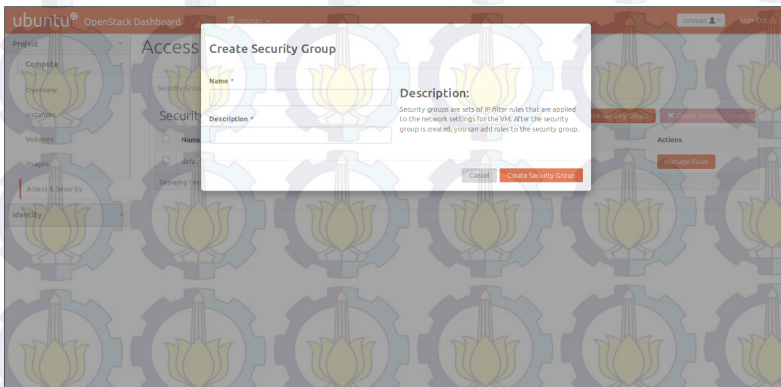


**Gambar 4.24:** Pengaturan *security group* melalui antarmuka

Pemberian izin agar mesin virtual dapat diakses melalui *remote* dapat dilakukan melalui antarmuka. Pengguna cukup memilih menu **Project**, kemudian dilanjutkan dengan **Compute** dan pilih menu **Access & Security**. Pada menu **Access & Security** dalam tab **Security Groups** terdapat sebuah tabel *security groups*. *Security groups* yang terdapat dalam tabel tersebut adalah **default**. Pemberian izin SSH, ICMP atau protokol TCP/IP lainnya dapat dilakukan dengan memilih menu **Manage Rules** dan dilanjutkan dengan **Add Rules**. Kemudian pengguna dapat memilih

SSH, ICMP atau protokol TCP/IP lainnya yang ingin dibuka atau diberi izin seperti Gambar 4.24.

Penambahan security group dapat dilakukan dengan memilih menu **Create Security Groups**, kemudian dilanjutkan dengan pengisian *form* yang disediakan seperti pada Gambar 4.25. Setelah penambahan selesai maka *security group* akan muncul pada tabelnya secara otomatis. Pengguna dapat menggunakannya saat pembuatan mesin virtual (*launch instance*), dan mengatur aturan perizinan setiap protokol seperti Gambar 4.24.



**Gambar 4.25:** Penambahan *security group* melalui antarmuka

#### 4.1.5 Uji Coba Cider (Block Storage Service)

Pengujian pada cinder dilakukan dengan melakukan pengecekan apakah seluruh layanan yang mendasari cinder sudah terpasang dan tidak menunjukkan status galat. Pengecekan dilakukan dengan menggunakan perintah `cinder service-list` pada *console* seperti pada Gambar 4.27.



```
rohman@controller:~$ nova list
```

ID	Name	Status	Task State	Power State	Networks
4d08ccfe-3a99-42f1-bc96-7acc4c11cc3	Ubuntu1	ACTIVE	-	Running	nova-net=192.168.100.130

```
rohman@controller:~$
```

Gambar 4.26: Melihat daftar mesin melalui *console*

```
rohman@controller:~$ cinder service-list
```

Binary	Host	Zone	Status	State	Updated at	Disabled Reason
cinder-scheduler	controller	nova	enabled	up	2015-05-10T16:04:30.000000	None
cinder-volume	controller	nova	enabled	up	2015-05-10T16:04:30.000000	None

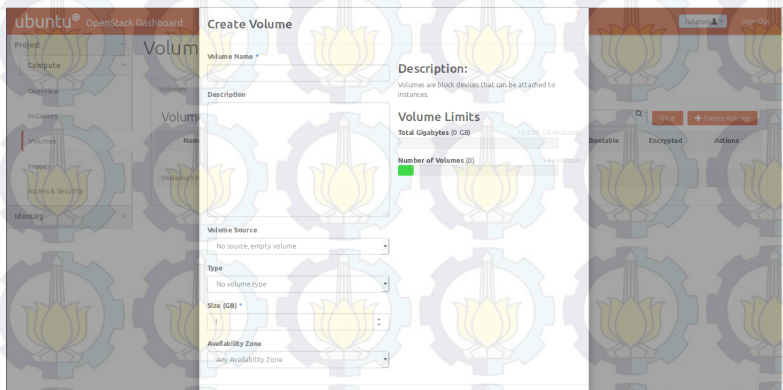
```
rohman@controller:~$
```

Gambar 4.27: Layanan-layanan pembangunan cinder

Di lanjutkan dengan pengujian dengan pembuatan penyimpanan (*volume*) tambahan untuk mesin virtual. Penggunaan perintah `cinder create` yang diikuti dengan penyimpanan dan ukurannya dengan satuan GB (*Gigabyte*) seperti pada Kode 4.10. Di mana nama dari penyimpanan yang dibuat adalah `Volume1` dengan kapasitas 1 GB.

```
$ cinder create --display-name Volume1 1
```

**Kode 4.10:** Perintah untuk menambah *volume* melalui *console*



**Gambar 4.28:** Penambahan *volume* melalui antarmuka

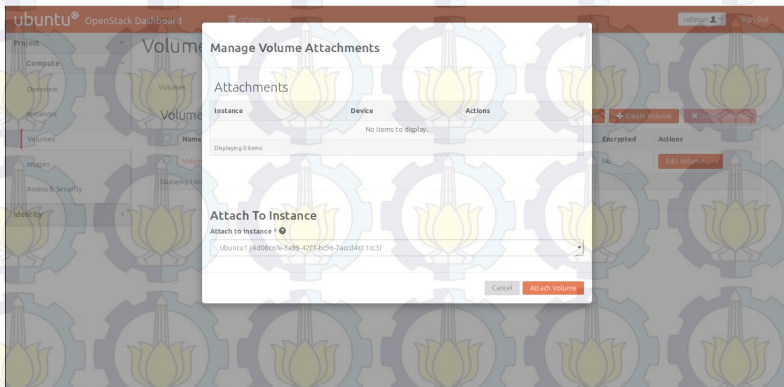
Pada penambahan *volume* melalui antarmuka, yang harus dilakukan pengguna adalah memilih **Project** dan dilanjutkan dengan **Compute**. Pilih **Volume** pada menu **Compute**, kemudian lanjutkan dengan **Create Volume** dan jangan lupa untuk mengisi *form* dan menu *dropdown* seperti pada Gambar 4.28. Pada pengujian ini, yang dilakukan adalah memasukkan nama *volume* dalam *form* nama penyimpanan dan *Size* untuk ukuran penyimpanan tambahan yang diinginkan. Proses tersebut sama dengan penggunaan perintah pada Kode 4.10

Penambahan penyimpanan yang telah dibuat ke mesin virtual juga dapat dilakukan dengan *console* dan juga antarmuka. Perintah pada *console* yang digunakan adalah `nova volume-attach`, yang

selanjutnya diikuti dengan nama mesin virtual dan nomor ID dari penyimpanan seperti pada Kode 4.11.

```
$ nova volume-attach Ubuntu1 \
158bea89-07db-4ac2-8115-66c0d6a4bb48
```

**Kode 4.11:** Perintah untuk menambah *volume* ke mesin virtual melalui *console*

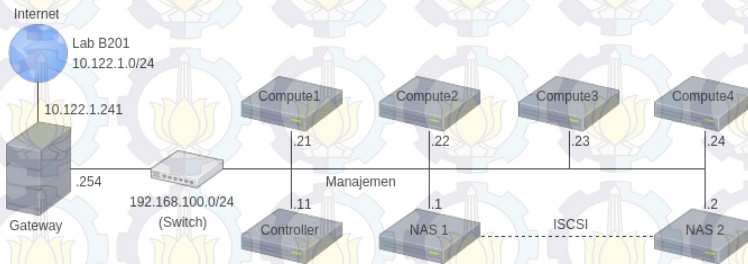


**Gambar 4.29:** Penambahan penyimpanan melalui antarmuka

Penambahan penyimpanan ke mesin virtual melalui antarmuka. Pengguna diharuskan untuk memilih **Project** kemudian **Compute**. Setelah memilih menu **Compute**, menu yang harus dipilih selanjutnya adalah **Volume**. Pada tabel penyimpanan terdapat sebuah kolom **Actions**, dan pada kolom tersebut terdapat *dropdown button*. Pada *dropdown button* kemudian pilih **Edit Attachments**, pilih mesin virtual yang ingin ditambah penyimpanannya pada menu *dropdown Attach to Instance* seperti pada Gambar 4.29. Status pada tabel penyimpanan akan berubah dan terdapat keterangan bahwa penyimpanan ditambahkan ke mesin virtual. Beracuan dengan proses penambahan penyimpanan ke mesin virtual melalui antarmuka dan console pada bagian ini, maka pengujian cinder dinyatakan berhasil.

## 4.2 Pengujian Topologi Jaringan

Terdapat dua jaringan yang terhubung pada *compute node* yaitu jaringan manajemen dan eksternal seperti pada Gambar 3.2. Pada awalnya jaringan yang digunakan hanya menggunakan manajemen, tanpa jaringan eksternal. Berawal hal tersebutlah pengujian dilakukan, penggunaan jaringan manajemen yang terhubung langsung melalui *gateway* seperti pada Gambar 4.30. Hasil dari penelitian tersebut gagal dan mesin virtual tidak bisa berjalan. Pada tabel daftar mesin virtual seperti pada Gambar 4.26, mesin virtual terus menunjukkan status *spawning* dalam jangka waktu lama dan kemudian berubah menjadi galat. Power state memberikan pesan *No State* dan alamat IP tidak muncul pada kolom *Networks Status*.



**Gambar 4.30:** Desain jaringan tanpa jaringan eksternal

Pada pengujian ini dapat ditarik kesimpulan bahwa mesin virtual pada OpenStack tidak dapat berjalan tanpa menggunakan jaringan eksternal. Di mana jaringan eksternal dari OpenStack terhubung dengan nova-network, yang berfungsi sebagai penyedia jaringan untuk mesin virtual yang berjalan pada setiap *compute node*.

## 4.3 Kemampuan Maksimal Komputasi Awan

Pengujian pada bagian ini bertujuan memperoleh data jumlah mesin virtual yang dapat dibuat oleh komputasi awan. Berapa batas spesifikasi yang dapat dicapai pada setiap mesin virtual juga dicari. Tiga elemen penting dari pengujian ini diantaranya komputasi (CPU), memory (RAM) dan penyimpanan. Ketiganya merupakan bagian penting dalam pembuatan mesin virtual pada sistem komputasi awan. Pada Tabel 3.1 dibagikan perangkat keras dan



sistem operasi menjelaskan spesifikasi dari *controller* dan *compute node*. Setiap komputasi pada sistem komputasi awan dijalankan pada *compute node*. Jumlah *compute node* yang digunakan sebanyak empat buah, ketika sumber daya dari setiap *node* dijumlah maka akan diperoleh spesifikasi seperti pada Tabel 4.1.

**Tabel 4.1:** Jumlah spesifikasi dari empat *compute node*

Spesifikasi	Keterangan
Jumlah Prosesor	8
Jumlah Core	32 core
Memori	24 GB
<i>Swap</i>	24 GB
<i>Disk Storage</i>	1200 GB

Setelah dilakukan pengujian batas maksimal dari komputasi awan, jumlah mesin virtual yang dapat dibuat bergantung pada jumlah memori. Semakin besar memori maka semakin banyak mesin virtual yang bisa dibuat, total memori yang dapat digunakan oleh pengguna memiliki batas perbandingan 1.5 : 1 dari jumlah memori pada setiap *compute node*. Kapasitas memori total dari *compute node* sebesar 24 GB, maka memori dari sistem komputasi awan sebesar 36 GB. Total memori sejumlah 36 GB tersebut diperoleh dari 24 GB memori dikali dengan 1.5. Pada komputasi jumlah maksimal core yang dapat dicapai sebesar 255 *core* dalam 1 mesin virtual, jumlah mesin virtual yang dapat dibuat menyesuaikan dengan jumlah memori. Sedangkan batas maksimal penyimpanan pada setiap mesin virtual adalah 2 GB.

#### 4.4 Perbandingan Performa Mesin Virtual

Pengujian bertujuan memperoleh data perbandingan performa antara mesin virtual layanan dari komputasi awan dengan sebuah komputer, keduanya menggunakan spesifikasi yang sama. Program *benchmark* yang digunakan adalah Phoronix Test Suite (PTS) dengan mode *interactive* dimana pengguna dapat memilih paket *benchmark* sesuai dengan kriteria yang ingin digunakan.

Seperti pada pengujian sebelumnya tiga elemen penting yang diukur dalam pengujian ini diantaranya komputasi (CPU), memory (RAM) dan penyimpanan (I/O). Selain itu dua pengujian lain juga turut ditambahkan pada bagian ini yaitu Apache Benchmark dan Compilation Benchmark. Kedua pengujian tersebut dilakukan karena terdapat hubungan antara CPU, memori dan penyimpanan dalam pemenuhan kebutuhan komputasi. Pada Tabel 4.2 merupakan spesifikasi (*flavor*) dari mesin virtual yang dibuat sama seperti spesifikasi komputer pada Tabel 4.3. Keduanya menggunakan sistem operasi Ubuntu 14.04 (Trusty Tahr) 64-bit.

**Tabel 4.2:** Spesifikasi (*flavor*) dari mesin virtual

Spesifikasi	Keterangan
Jumlah Prosesor	1
Jumlah Core	4 Core
Memori	8 GB
Swap	8 GB
<i>Disk Storage</i>	160 GB

**Tabel 4.3:** Spesifikasi dari komputer yang akan dibandingkan

Spesifikasi	Keterangan
Jumlah Prosesor	1
Jumlah Core	4 Core
Jenis Prosesor	AMD A8-3850 APU
Memori	8 GB
Swap	8 GB
<i>Disk Storage</i>	160 GB

Selain membandingkan performa antara mesin virtual yang merupakan layanan dari komputasi awan dengan sebuah komputer. Di bandingkan juga performa dari mesin virtual dengan spesifikasi yang terus dinaikan berdasarkan spesifikasinya. Pada pengujian ini yang diuji adalah performa dari memori dan komputasi (CPU)

saja. Pada Tabel 4.4 merupakan *flavor* atau spesifikasi dari mesin virtual dengan jumlah *core* yang terus dinaikan, hal ini bertujuan untuk menguji performa dari komputasi (CPU). Sedangkan pada Tabel 4.5 merupakan *flavor* atau spesifikasi dari mesin virtual dengan jumlah memori (RAM) yang terus dinaikan, hal ini bertujuan untuk menguji performa dari memori (RAM) dari mesin virtual. Melalui pengujian tersebut dapat terukur berapa kenaikan performa mesin virtual setiap kenaikan spesifikasinya.

**Tabel 4.4:** *Flavor* mesin virtual untuk mengukur performa komputasi.

Spec.	m3.test1	test2	test3	test4	test5
Processor	1	1	1	1	1
Core	1	2	4	8	16
Memori	2 GB	2 GB	2 GB	2 GB	2 GB
Swap	1 GB	1 GB	1 GB	1 GB	1 GB
Storage	160 GB	160 GB	160 GB	160 GB	160 GB

**Tabel 4.5:** *Flavor* mesin virtual untuk mengukur performa memori.

Spec.	test1	test2	test3	test4	test5
Processor	1	1	1	1	1
Core	4	4	4	4	4
Memori	512 MB	1 GB	2 GB	4 GB	8 GB
Swap	256 MB	512 MB	1 GB	2 GB	4 GB
Storage	160 GB	160 GB	160 GB	160 GB	160 GB

#### 4.4.1 Performa Komputasi (CPU)

Pengujian kinerja CPU secara keseluruhan dilakukan dengan mengukur kecepatan pemrosesan beban kerja. Melalui pengujian tersebut maka dapat ditarik sebuah standar perbandingan performa antara mesin virtual layanan dari komputasi awan dengan sebuah komputer. Tes profil dari Phoronix Test Suite yang digunakan dalam pengujian ini diantaranya C-ray, Stream, dan FFMpeg.

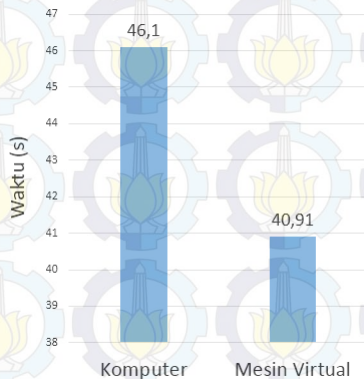
## C-Ray

C-Ray adalah sebuah program *raytracer* sederhana yang bertujuan untuk menguji kemampuan *floating-point* dari CPU. Pada tes ini akan dibuat 16 threads yang akan berjalan pada setiap core untuk menjalankan sebuah proses. Pada proses tersebut dilakukan 8 kali penembakan pada setiap piksel untuk proses *anti aliasing* (penghalusan gambar) dan akan dihasilkan sebuah gambar dengan ukuran 1600 x 1200 [7].

Tes ini hanya membutuhkan sedikit data dan tidak menggunakan RAM. Hanya CPU yang diuji dan hasilnya tidak dipengaruhi oleh komponen lain dari komputer atau mesin virtual. Hal tersebut yang menjadikan pengujian ini ideal untuk menguji kecepatan komputer.

**Tabel 4.6:** Waktu pemrosesan beban kerja antara mesin virtual dan komputer

Perangkat	Waktu (s)
Komputer	46,1
Mesin Virtual (VM)	40,91



**Gambar 4.31:** Perbedaan waktu pemrosesan beban kerja antara mesin virtual dengan komputer



Tolok ukur pada pengujian ini adalah waktu untuk memproses beban kerja. Semakin sedikit angka yang muncul pada hasil *benchmark* berarti semakin sedikit waktu yang dibutuhkan untuk memproses beban kerja. Pada Tabel 4.6 adalah hasil perbandingan antara mesin virtual dengan spesifikasi seperti pada Tabel 4.2 dan komputer yang spesifikasinya dijelaskan pada Tabel 4.3. Di mana kecepatan dari mesin virtual dalam memproses beban tersebut sekitar 40,91 detik, sedangkan komputer 46,10 detik. Hasil perbandingan tersebut juga dapat dilihat grafik perbandingan pada Gambar 4.31.

Semakin banyak penggunaan *thread* maka akan semakin mempercepat proses, tentunya penggunaan jumlah *thread* memiliki batas. Ketika penambahan *thread* terus dilakukan seperti pada pengujian ini, tetapi kecepatan mengolah beban sudah tidak bertambah. Maka dapat disimpulkan itu adalah kecepatan maksimal dari processor.

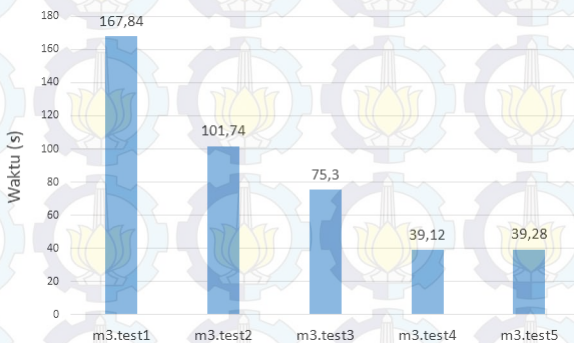
Selain itu diukur juga waktu dari mesin virtual dalam memproses beban kerja dengan spesifikasi yang terus dinaikan seperti pada Tabel 4.4 dengan menggunakan C-Ray. Spesifikasi yang digunakan seperti pada Tabel 4.4 dikarenakan fokus dari pengujian ini untuk mengukur kemampuan dari komputasi. Hasil dari pengujian ini dapat dilihat pada Tabel 4.7, dimana spesifikasi dari test1 yang kemudian ditambahkan sebuah *core* menjadi test2 terdapat pengurangan waktu sebesar 66,1 s. Sedangkan test2 ke test3 dimana terjadi penambahan *core* sejumlah 2 *core*, hal tersebut mempersingkat waktu pemrosesan sebesar 26,4 s. Pada test3 ke test4 terjadi penambahan *core* sejumlah 4 *core*, kemudian waktu pemrosesan berkurang lagi sebesar 36,16 s. Rata-rata waktu pemrosesan beban berkurang sebesar 18,41 s per-*core*.

Penambahan *core* pada test4 ke test5 sejumlah 8 *core*, tidak terlalu berpengaruh pada pemrosesan beban yang diuji dengan menggunakan C-Ray, perbandingan tersebut dapat dilihat pada Gambar 4.32. Kecepatan justru turun sebesar 18 s seperti pada Tabel 4.7, hal ini disebabkan oleh jumlah *core* dari *compute node* yang menjadi induk tempat berjalannya mesin virtual berjumlah lebih dari 8 *core*. Jika jumlah *core* melebihi dari 8 *core*, maka *core* tersebut merupakan *core* virtual dari OpenStack yang diperoleh

dari *compute node* lainnya. Performa *core* tambahan dari *compute node* lain tersebut berbeda jauh dengan performa dari *core* yang diperoleh dari *compute node* tempat mesin itu berjalan. Hal tersebut dikarenakan mesin virtual langsung melakukan akses ke perangkat keras melalui layer virtualisasi seperti dijelaskan pada bagian virtualisasi pada bab dasar teori.

**Tabel 4.7:** Waktu pemrosesan beban kerja pada mesin virtual

Flavor	Waktu (s)
test1	167,84
test2	101,74
test3	75,3
test4	39,12
test5	39,28



**Gambar 4.32:** Perbedaan waktu pemrosesan beban kerja pada mesin virtual

## Stream

*Stream* adalah sebuah patokan untuk mengukur memori *bandwidth*. Memori *bandwidth* dapat diartikan seberapa cepat data dapat ditulis atau dibaca dari memori oleh *processor*. Hal ini juga mempengaruhi seberapa cepat sistem operasi bisa memperoleh da-

ta dari memori untuk diproses. Jika bandwidth memori rendah, maka prosesor harus menunggu terlebih dahulu dalam setiap proses pengambilan dan penulisan data. Jika bandwidth memori tinggi, maka proses pengambilan dan penulisan data dapat dilakukan dengan cepat [8].

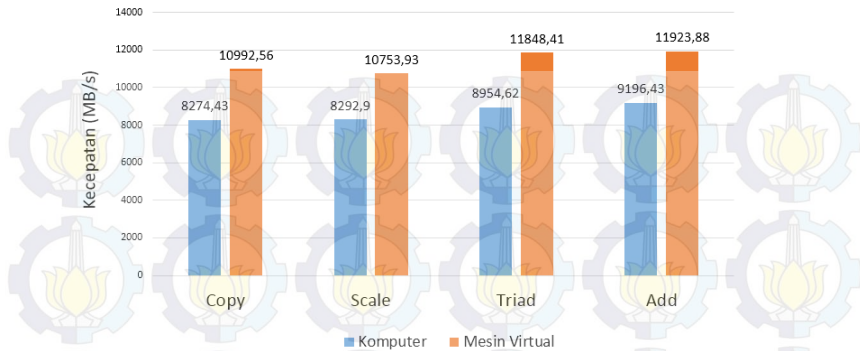
Pengukuran kecepatan dari stream terdiri dari empat kernel vektor sederhana yaitu *Copy*, *Scale*, *Add* dan *Triad*, satuan dari stream atau pengukuran memori bandwidth adalah MB/s [9]. Pada Tabel 4.8 adalah perbandingan hasil *stream benchmark*, semakin tinggi nilai benchmark maka semakin besar ukuran memori bandwidth. Pada komputer diperoleh kecepatan 8274,43 MB/s untuk *copy*, 8292,9 MB/s untuk *scale*, 8954,62 MB/s untuk *Triad* dan 9196,43 untuk *Add*. Sedangkan pada mesin virtual diperoleh kecepatan 10992,56 MB/s untuk *copy*, 10753,93 untuk *scale*, 11848,41 MB/s untuk *Triad* dan 11848,41 MB/s untuk *Add*. Hasil perbandingan tersebut juga dapat dilihat grafik perbandingan pada Gambar 4.33.

**Tabel 4.8:** Hasil pengujian memori *bandwidth* pada komputer dan mesin virtual

Perangkat	Add	Copy	Scale	Triad
Komputer	9196,43	8274,43	8292,9	8954,62
Mesin Virtual (VM)	11758,46	10992,56	10753,93	11923,88

Di ukur juga bandwidth dari mesin virtual dengan spesifikasi yang terus dinaikan seperti pada Tabel 4.4 dengan menggunakan Stream. Spesifikasi yang digunakan tetap seperti pada Tabel 4.4 dikarenakan fokus dari pengujian ini untuk mengukur memori bandwidth, seberapa cepat data dapat ditulis atau dibaca dari memori oleh *processor*.

Hasil dari pengujian ini dapat dilihat pada Tabel 4.16, dimana spesifikasi dari mesin virtual test1 yang kemudian ditambah jumlah *core* sebanyak 1 menjadi test2 terdapat penambahan kecepatan sebesar 1789,01 MB/s untuk proses *copy*, 1550,4 MB/s untuk *scale*, 902,33 MB/s untuk *triad* dan 871,73 MB/s untuk *add*. Sedangkan test2 ke test3 dimana terjadi penambahan *core* sejumlah 2 *core*,



**Gambar 4.33:** Perbandingan memori bandwidth antara mesin virtual dengan komputer

kecepatan mesin virtual bertambah sebesar 385,58 MB/s untuk proses *copy*, 682,27 MB/s untuk *scale*, 1584,79 MB/s untuk *triad* dan 1609,72 MB/s untuk *add*. Pada test3 ke test4 terjadi penambahan *core* sejumlah 4 *core*, kemudian kecepatan mesin virtual bertambah sebesar 936,72 MB/s untuk proses *copy*, 678,9 MB/s untuk *scale*, 665,66 MB/s untuk *triad* dan 615,03 MB/s untuk *add*. Rata-rata kecepatan dari mesin virtual bertambah sebesar 444,5 MB/s untuk proses *copy*, 415,93 MB/s untuk *scale*, 450,39 MB/s untuk *triad* dan 442,35 MB/s untuk *add* per-*core*.

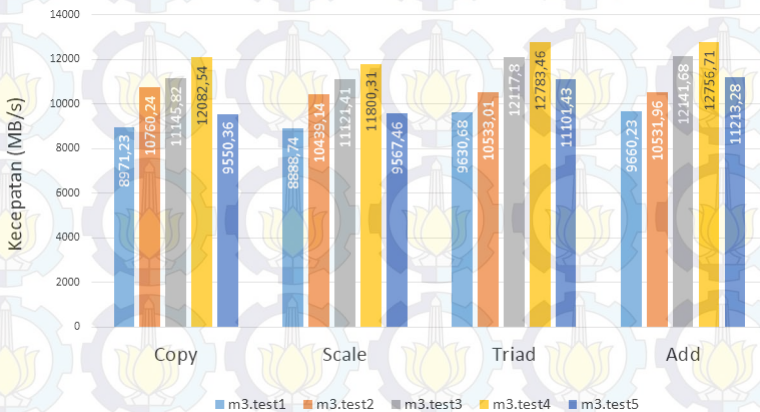
**Tabel 4.9:** Hasil pengujian performa memori *bandwidth* pada mesin virtual

Flavor	Add	Copy	Scale	Triad
test1	8971,23	8888,74	9630.68	9660.23
test2	11760.24	11439.14	12533.01	12531.96
test3	11145.82	11121.41	12117.8	12141.68
test4	12082.54	11800.31	12783.46	12756.71
test5	9550.36	9567.46	11101.43	11213.28

Penambahan *core* pada mesin virtual test4 ke test5 sejumlah 8 *core*, tidak terlalu berpengaruh pada pemrosesan beban yang diuji dengan menggunakan C-Ray, perbandingan tersebut dapat dilihat



pada Gambar 4.34. Kecepatan justru turun sebesar 2532,18 MB/s untuk proses *copy*, 2232,85 MB/s untuk *scale*, 1682,03 MB/s untuk *triad* dan 1543,43 MB/s untuk *add* seperti pada Tabel 4.16, hal ini disebabkan oleh jumlah core dari *compute node* yang menjadi induk tempat berjalannya mesin virtual berjumlah 8 core. Seperti penjelasan pada pengujian dengan menggunakan C-Ray. Jika core melebihi dari 8 core, maka core tersebut merupakan core virtual dari OpenStack yang diperoleh dari *compute node* lainnya. Performa *core* tambahan dari *compute node* lain tersebut berbeda jauh dengan performa dari *core* yang diperoleh dari *compute node* tempat mesin itu berjalan. Hal tersebut disebabkan mesin virtual langsung melakukan akses ke perangkat keras melalui layer virtualisasi seperti dijelaskan pada bagian virtualisasi pada bab dasar teori.



**Gambar 4.34:** Perbedaan memori bandwidth pada mesin virtual

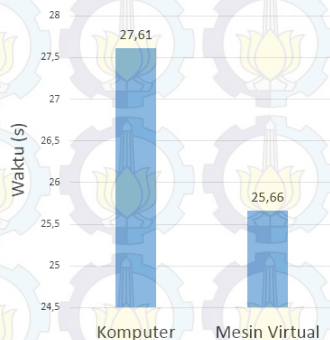
## FFmpeg

Pengujian performa *encoding* dengan menggunakan FFmpeg *encoding utility* yang disediakan oleh Phoronix Test Suite. Proses ini memakan RAM dan penyimpanan saat melakukan proses video *encoding*, tetapi saat melakukan pemrosesan *file* yang sangat besar, kunci utama dari proses ini adalah kemampuan CPU. Maka dari itu dibutuhkan kemampuan CPU yang cepat untuk konversi video [10].

Fakta tersebutlah yang mendasari perlunya pengujian ini dilakukan. Hasil perbandingan performa dapat dilihat pada Tabel 4.10, dimana kemampuan mesin virtual yang merupakan layanan dari komputasi awan mampu melakukan proses encoding dengan kecepatan 25,66 s. Sedangkan pada komputer, waktu yang dibutuhkankannya adalah 27,61 *encoding*. Hasil perbandingan tersebut juga dapat dilihat grafik perbandingan pada Gambar 4.35.

**Tabel 4.10:** Hasil pengujian kecepatan proses encoding

Perangkat	Waktu (s)
Komputer	27,61
Mesin Virtual (VM)	25,66



**Gambar 4.35:** Perbandingan kecepatan encoding

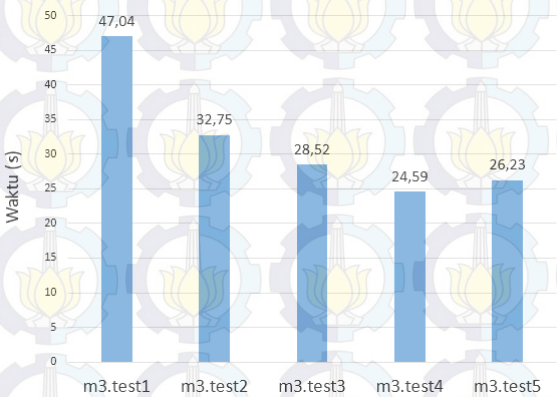
Selain itu diukur juga waktu dari mesin virtual dalam melakukan proses *encoding* dengan spesifikasi yang terus dinaikan seperti pada Tabel 4.4 dengan menggunakan FFmpeg *benchmark*. Hasil dari pengujian ini dapat dilihat pada Tabel 4.17, dimana spesifikasi dari test1 yang kemudian ditambahkan sebuah *core* menjadi test2 terdapat pengurangan waktu sebesar 14,29 s. Sedangkan test2 ke test3 dimana terjadi penambahan *core* sejumlah 2 *core*, hal tersebut mempersingkat waktu pemrosesan sebesar 4,23 s. Pada test3 ke test4 terjadi penambahan *core* sejumlah 4 *core*, kemudian wak-

tu pemrosesan berkurang lagi sejumlah 3,93 s. Rata-rata waktu pemrosesan beban berkurang sebesar 3,2 s per-core.

Penambahan *core* pada test4 ke test5 sejumlah 8 *core*, juga tidak terlalu berpengaruh pada proses *encoding* yang diuji dengan menggunakan FFMpeg *benchmark*, perbandingan tersebut dapat dilihat pada Gambar 4.41. Di mana waktu bertambah sebesar 1,64 s seperti pada Tabel 4.17, hal ini disebabkan oleh jumlah *core* dari *compute node* yang menjadi induk tempat berjalannya mesin virtual berjumlah lebih dari 8 *core*.

**Tabel 4.11:** Waktu yang diperlukan saat *encoding*

Flavor	Waktu (s)
test1	47,04
test2	32,75
test3	28,52
test4	24,59
test5	26,23



**Gambar 4.36:** Perbedaan waktu saat *encoding*

Seperti penjelasan pada pengujian dengan menggunakan C-Ray dan Stream. Jika jumlah *core* melebihi dari 8 *core*, maka *core* ter-

sebut merupakan *core virtual* dari OpenStack yang diperoleh dari *compute node* lainnya. Performa *core* tambahan dari *compute node* lain tersebut berbeda jauh dengan performa dari *core* yang diperoleh dari *compute node* tempat mesin itu berjalan. Hal tersebut dikarenakan mesin virtual langsung melakukan akses ke perangkat keras melalui layer virtualisasi seperti dijelaskan pada bagian virtualisasi pada bab dasar teori.

#### 4.4.2 Performa Penyimpanan

Penyimpanan bukan hanya dipengaruhi oleh besarnya kapasitas *hard disk drive* untuk menyimpan data. Performa dari penyimpanan tersebut juga berpengaruh dalam menangani setiap proses transaksi data, baik data itu diakses secara langsung oleh pengguna atau diakses secara otomatis oleh program. Tiga faktor yang berpengaruh pada performa penyimpanan diantaranya adalah *Throughput*, *IOPS (Input/Output Operations per Seconds)* dan *Latensi (Delay)*.

Maksud dari *Throughput* adalah berapa banyak data atau informasi yang dapat diakses dalam satuan waktu (MB/s), sedangkan *IOPS (Input/Output Operations per Seconds)* adalah masukan atau keluaran yang dapat dilakukan oleh penyimpanan dalam satuan detik. Kemudian untuk latensi secara umum adalah waktu saat melakukan permintaan pada komputer dan menerima jawabannya, sedangkan lebih spesifiknya pada penyimpanan dapat diartikan sebagai waktu pemilihan sektor untuk dibaca atau ditulis.

Tes profil yang digunakan dalam pengujian ini diantaranya *aio-stress*, *fio*, *dbench*, dan *postmark*. Berikut merupakan penjelasan tes dan perbandingan hasil *benchmark* pada penyimpanan.

##### Aio-stress

Tes ini memaksa penyimpanan untuk bekerja sesuai dengan kemampuan input dan outputnya secara tidak sinkron sampai batas kemampuannya. Tes ini menggunakan *file* berukuran 2048 MB yang terbagi menjadi 64 KB setiap bagiannya [7]. Setelah melakukan pengujian ini maka akan didapat *bandwidth* dari penyimpanan.

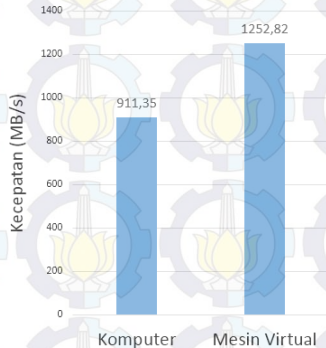
Hasil perbandingan performa dapat dilihat pada Tabel 4.7, dimana kemampuan mesin virtual yang merupakan layanan dari



komputasi awan memiliki *bandwidth* dan *throughput* lebih besar dari pada komputer. Di mana mesin virtual mampu memncapai kecepatan 1252,82 MB/s dan komputer mencapai 911,35 MB/s. Hasil perbandingan tersebut juga dapat dilihat grafik perbandingan pada Gambar 4.37.

**Tabel 4.12:** Hasil pengujian penyimpanan dengan Aio-Stress

Perangkat	Informasi per detik
Komputer	911,35
Mesin Virtual (VM)	1252,82



**Gambar 4.37:** Perbandingan bandwidth penyimpanan

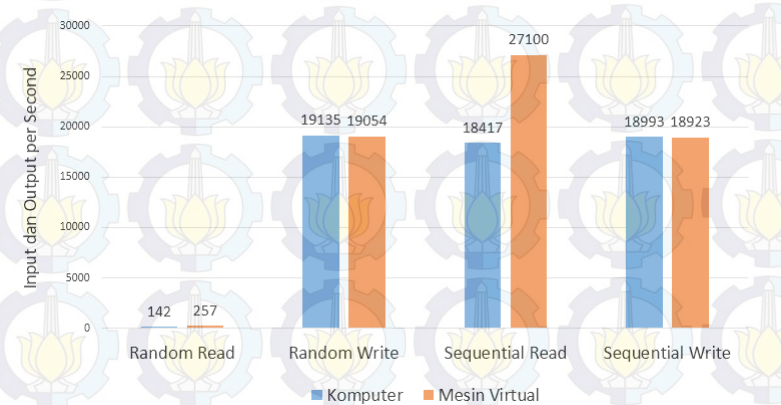
## Fio

Fio adalah kepanjangan dari *Flexible I/O Tester*. Tes ini akan memberikan beban kerja berupa input dan output pada penyimpanan. Pemberian beban tersebut berupa proses I/O yang dilakukan secara acak dan juga berurutan, selain itu proses pembacaan dan penulisan setiap beban juga turut di uji [10]. Tes ini menjalankan sebuah program yang mengakses penyimpanan dari mesin virtual dan komputer. Data-data yang diakses berukuran sangat kecil, rata-rata berukuran kurang dari 4 KB. Hal tersebut bertujuan mengukur kecepatan IOPS (*Input/Output Operations per Second*).

Hasil perbandingan performa dapat dilihat pada Gambar 4.38, menunjukkan kemampuan *input* atau *output* (I/O) dari mesin virtual dan komputer. Mesin virtual mampu menerima 275 *input* atau *output* melalui pengujian membaca secara acak (*random read*) dalam satu detik, sedangkan komputer hanya mampu menerima 142 I/O dalam setiap detiknya.

Tabel 4.13: Hasil pengujian penyimpanan dengan FIO

Perangkat	Random Read	Random Write	Sequential Read	Sequential Write
Komputer	142	19135	18417	18993
VM	257	19054	27100	18923



Gambar 4.38: Perbandingan I/O penyimpanan dengan FIO

Mengacu pada hasil pengujian pada Tabel 4.13, Di mana kemampuan penyimpanan dari mesin virtual dan komputer kembali diuji dengan tes penulisan secara acak (*random write*), mesin virtual memperoleh skor 19054 sedangkan komputer memperoleh skor 19135. Selain diuji dengan cara random atau acak, mesin virtual dan komputer diuji juga dengan cara yang urut. Pada pengujian membaca secara urut (*sequential read*) mesin virtual memperoleh nilai 27100, sedangkan komputer memperoleh nilai 18417. Di lan-

jutkan dengan pengujian penulisan secara urut (*sequential write*), maka diperoleh skor 18993 dari mesin virtual dan skor 18993 dari komputer. Perbandingan tersebut juga dapat dilihat pada Gambar 4.38. Nilai-nilai tersebut menunjukkan jumlah maksimal input atau output (I/O) yang bisa dilakukan dalam satuan detik.

#### 4.4.3 Performa Memori (RAM)

Sudah menjadi hal yang umum jika memori sangat berpengaruh pada pemrosesan beban kerja dan kecepatan dari komputer. Selain itu kemampuan RAM tidak hanya bergantung pada besarnya kapasitas memori, tetapi kecepatan memori, bandwidth dan cache juga memiliki pengaruh yang tidak kalah pentingnya.

Tes profil yang digunakan dalam pengujian ini diantaranya RAMspeed dan cachebench. Sedangkan pengujian pengukuran *bandwidth* pada memori sudah dilakukan pada bagian performa komputer (CPU) seperti yang ditampilkan pada Gambar 4.33.

##### RAMspeed

Hasil dari pengujian dengan menggunakan profil RAMspeed berupa *throughput* yang ditampilkan dalam satuan MB/s. Semakin tinggi angka hasil benchmark maka semakin baik performansinya. Terdapat dua nilai penting pada tes tersebut diantaranya *floating-point* dan *integer*. Selain itu terdapat juga empat metode pengujian yang berperan penting pada pengujian ini seperti add, copy, scale dan triad yang dijelaskan sebelumnya pada bagian performa komputer (CPU).

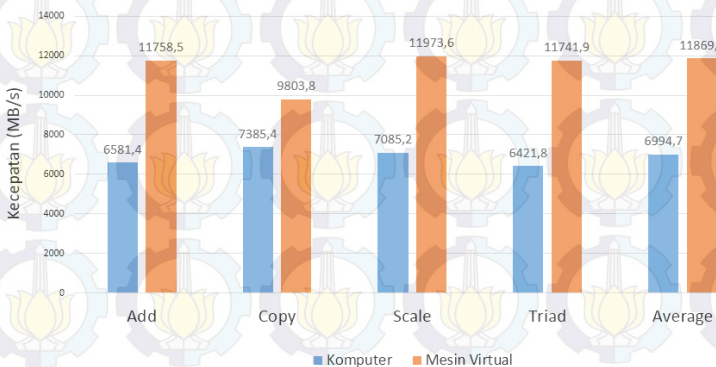
Pada Tabel 4.14 dapat dilihat perbandingan *throughput* antara mesin virtual dengan komputer saat memproses data integer dan pada Table 4.15 merupakan perbandingan saat memproses *floating-point*. Mesin virtual mampu mencapai 11869.96 MB/s untuk ngolah data integer dan memperoleh *throughput* 12104.3 MB/s untuk *floating point*. Sedangkan *throughput* dari komputer dapat mencapai 6994.73 MB/s untuk mengolah data *integer* dan 7631.58 MB/s untuk *floating point*. Perbandingan tersebut juga dapat dilihat pada Gambar 4.39 untuk pemrosesan data *integer* dan pada Gambar 4.39 untuk *floating point*.

**Tabel 4.14:** Hasil pengujian *throughput* memori saat mengolah data *integer* pada mesin virtual dan komputer

Perangkat	Add	Copy	Scale	Triad	Average
Komputer	6581,44	7385,36	7085,23	6421,75	6994,73
VM	11758,46	9803,82	11973,57	11741,93	11869,96

**Tabel 4.15:** Hasil pengujian *throughput* memori saat mengolah *floating-point* pada mesin virtual dan komputer

Perangkat	Add	Copy	Scale	Triad	Average
Komputer	7602,5	7366,8	7161,3	7835,6	7631,6
VM	12825,1	11527	11548,7	12799,8	12104,3



**Gambar 4.39:** Perbandingan *throughput* saat memproses *integer* pada komputer dan mesin virtual

Di ukur juga kecepatan memori dari mesin virtual dengan spesifikasi yang terus dinaikan seperti pada Tabel 4.5 dengan menggunakan RAMspeed. Spesifikasi yang digunakan tetap seperti pada Tabel 4.5 dikarenakan fokus dari pengujian ini untuk mengukur *throughput* dari memori, seberapa cepat memori dapat memproses data *integer* dan *floating-point*. Hasil dari pengujian ini dapat dilihat pada Tabel 4.16 yang merupakan perbandingan *throughput*



pada mesin virtual saat mengolah data integer dan pada Tabel 4.17 adalah saat pemrosesan data *floating-point*. Keduanya dibandingkan dengan cara menaikkan jumlah memorinya, kemudian ditampilkan perbandingannya pada Gambar 4.41 dan Gambar 4.42.

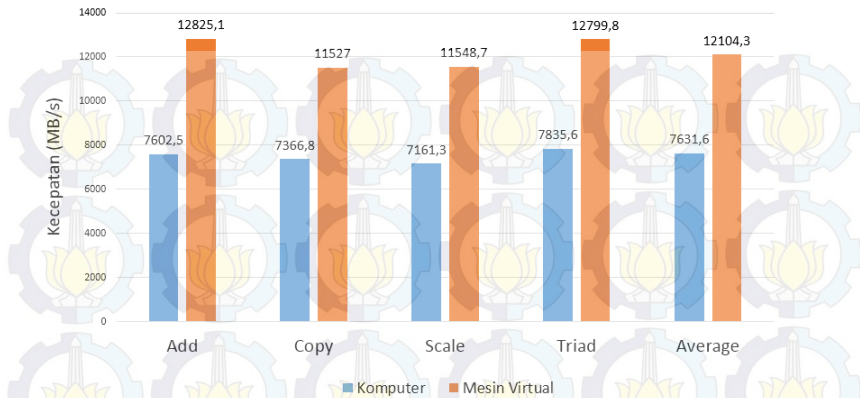
**Tabel 4.16:** Hasil pengujian *throughput* saat memproses data *integer* pada mesin virtual

Flavor	Add	Copy	Scale	Triad	Average
test1	4295,32	2992,39	2893,51	2910,12	2906,54
test2	3548,67	3072,18	2952,02	2956,76	2993,99
test3	3061,82	2982,32	3013,37	2986,03	3025,12
test4	3096,77	2975,57	2977,99	2990,69	3132,68
test5	11169,13	11564,66	11629,49	11042,92	11391,08

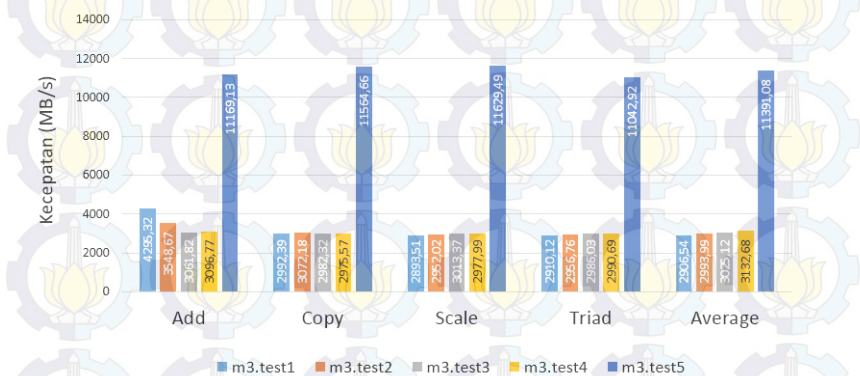
**Tabel 4.17:** Hasil pengujian *throughput* saat memproses *floating-point* pada mesin virtual

Flavor	Add	Copy	Scale	Triad	Average
test1	3090,08	2989,48	2907,53	3328,14	3223,74
test2	3263,81	2968,35	3082,33	3357,2	3649,7
test3	3453.79	2980.2	2989.58	3382,68	4583,37
test4	3333.98	2991.91	3142.49	3257.95	4441.79
test5	12291.92	11357.5	10072.47	12315.75	11892.81

Data perbandingan diperoleh dari penambahan spesifikasi dari mesin virtual test1 yang kemudian ditambah jumlah memorinya sebesar 512 MB menjadi test2 terdapat penambahan kecepatan sebesar 87.4 MB/s dalam mengolah data *integer* dan 425.96 MB/s untuk 425.96 MB/s untuk *floating-point*. Sedangkan test2 ke test3 dimana terjadi menambahkan memori sebesar 1 GB, kecepatan mesin virtual bertambah sebesar 31.13 MB/s untuk *integer*, dan 933.67 MB/s untuk *floating-point*.



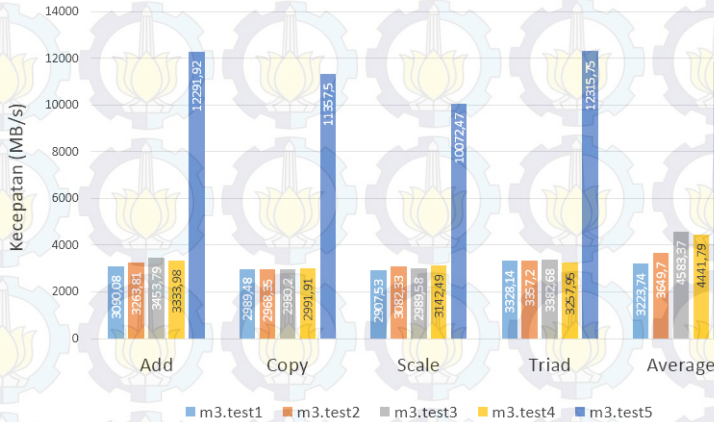
**Gambar 4.40:** Perbandingan *throughput* saat memproses *floating point* antara mesin virtual dengan komputer



**Gambar 4.41:** Perbedaan *throughput* saat memproses data *integer* pada mesin virtual.

Pada test3 ke test4 terjadi penambahan memori sebesar 2 GB, kemudian kecepatan mesin virtual bertambah sebesar 107.56 MB/s untuk integer dan sedangkan untuk floating-point mengalami penurunan kecepatan sebesar 141.58 MB/s dari pengujian sebelumnya pada *flavor* test test3. Sedangkan pada test4 ke test5 dimana terjadi penambahan memori sebesar 4 GB, kecepatan mesin virtual ber-

tambah sebesar 8258.4 MB/s untuk pemrosesan data *integer*, dan 7451.02 MB/s untuk *floating-point*. Rata-rata kecepatan dari mesin virtual bertambah sebesar 1205,83 MB/s untuk pemrosesan data *integer* dan 1208,01 MB/s untuk pemrosesan *floating-point* setiap penambahan 1 GB memori.



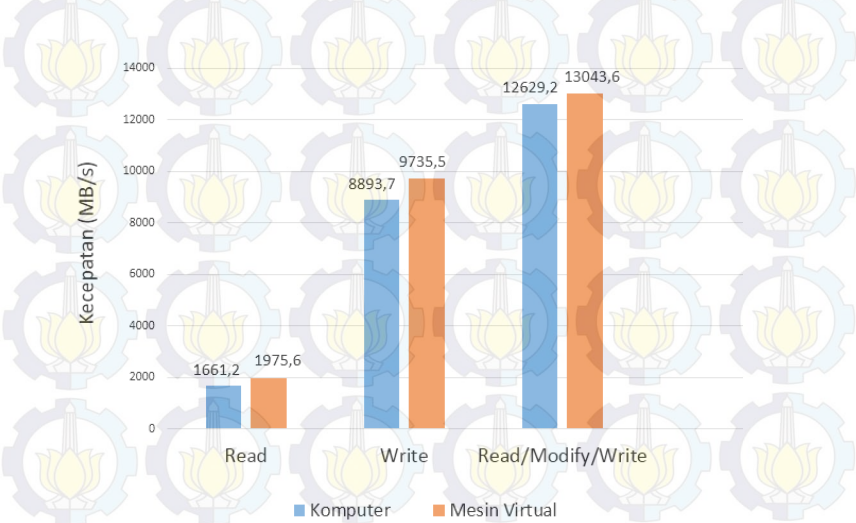
**Gambar 4.42:** Perbedaan *throughput* saat memproses *floating-point* pada mesin virtual

## Cachebench

CacheBench dirancang untuk menguji performa memori dan bandwidth cache [7]. Pada Tabel 4.18 merupakan hasil perbandingan performa memori cache yang terdapat pada mesin virtual dan komputer. Di mana saat proses membaca (*read*) mesin virtual dapat mencapai kecepatan 1975.64 MB/s. Kemudian untuk menulis (*write*) mesin virtual mampu mencapai 9735.51 MB/s dan untuk proses Read/Modify/Write dapat mencapai 13043.6 MB/s seperti pada Tabel 4.18. Pada komputer proses membaca dapat mencapai kecepatan 1661.15 MB/s, 8893.67 MB/s untuk proses menulis dan 12629.21 MB/s untuk proses *read/write/modify*. Hasil perbandingan tersebut juga dapat dilihat grafik perbandingan pada Gambar 4.43.

**Tabel 4.18:** Hasil perbandingan kecepatan *cache* memori antara mesin virtual dengan komputer

Perangkat	Read	Write	Read/Modify/Write
Komputer	277	35153	18947
Mesin Virtual (VM)	917	19054	27327



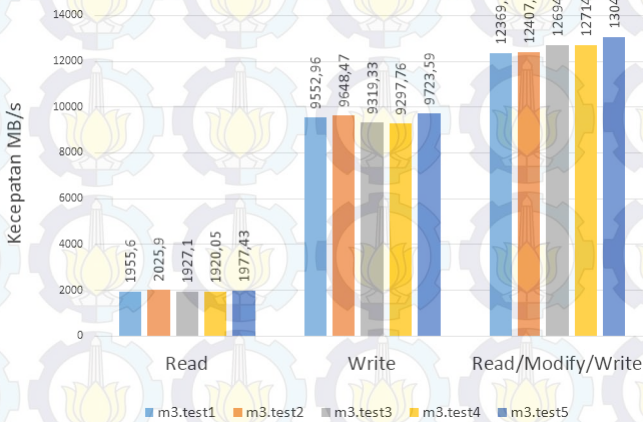
**Gambar 4.43:** Perbandingan kecepatan *cache* memori antara mesin virtual dengan komputer

Selain itu diukur juga kecepatan *cache* memori dari mesin virtual dengan spesifikasi yang terus dinaikan seperti pada Tabel 4.5 dengan menggunakan CacheBench. Spesifikasi yang digunakan tetap seperti pada Tabel 4.5 dikarenakan fokus dari pengujian ini untuk mengukur bandwidth dari *cache* memori, seberapa cepat *cache* memori dalam melakukan proses baca, tulis dan *read/write/modify*. Hasil dari pengujian ini dapat dilihat pada Tabel 4.17 yang merupakan perbandingan *bandwidth* pada mesin virtual saat menjalankan proses baca, tulis dan *read/write/modify*. Keduanya dibandingkan dengan cara menaikkan jumlah memorinya, hasil perbandingannya dapat dilihat pada Gambar 4.44.



**Tabel 4.19:** Hasil perbandingan *bandwidth cache* memori pada mesin virtual

Flavor	Read	Write	Read/Modify/Write
test1	1955,6	9552,96	12369,82
test2	2025,9	9648,47	12407,47
test3	1927,1	9319,33	12694,97
test4	1920,05	9297,76	12714,6
test2	1977,43	9723,59	13042,49



**Gambar 4.44:** Perbedaan *bandwidth cache* memori mesin virtual

Data tersebut diperoleh dari penambahan spesifikasi dari mesin virtual test1 yang kemudian ditambah jumlah memorinya sebesar 512 MB menjadi test2 terdapat penambahan kecepatan sebesar 70.3 MB/s saat proses membaca, 95.51 MB/s saat menulis dan 37.65 MB/s saat proses *read/write/modify*. Sedangkan test2 ke test3 dimana terjadi penurunan kecepatan sebesar 98.8 MB/s saat proses membaca, penurunan sebesar 329.14 MB/s juga terjadi saat menulis, akan tetapi terjadi kenaikan sebesar 287.5 MB/s saat proses *read/write/modify*.

Pada test3 ke test4 dilakukan penambahan memori sebesar 2 GB, kemudian kecepatan mesin virtual justru semakin berkurang sebesar 7.05 MB/s saat proses membaca, berkurang sebesar 21.56 MB/s saat menulis, akan tetapi bertambah 19.63 saat proses *read/write/modify*. Sedangkan pada test4 ke test5 dimana dilakukan penambahan memori sebesar 4 GB, kecepatan mesin virtual bertambah sebesar 57.38 MB/s untuk proses membaca, bertambah sebesar 425.83 MB/s untuk proses menulis dan bertambah sebesar 327.88 MB/s saat proses *read/write/modify*. Rata-rata kecepatan dari mesin virtual berkurang sebesar 1,9 MB/s untuk proses baca, bertambah sebesar 38,02 MB/s saat proses tulis dan bertambah sebesar 93,4 MB/s saat proses *read/write/modify* setiap penambahan 1 GB memori.

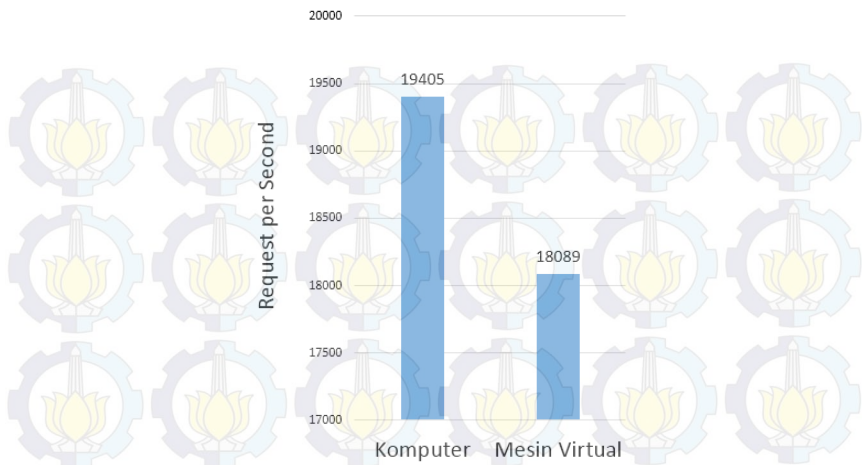
#### 4.4.4 Apache Benchmark

Pada pengujian ini, komputer dan mesin virtual diberi tugas sebagai *web server*. Beban yang akan diberikan pada komputer atau mesin virtual berupa permintaan sejumlah 100 sampai dengan 1.000.000 setiap detiknya [7]. Di desain untuk memberikan gambaran performa dari *webserver*, secara khusus akan menampilkan seberapa banyak request per detik atau seberapa banyak request per detik yang bisa dilayani mesin virtual atau komputer.

Pada Tabel 4.20 merupakan perbandingan performa antara mesin virtual dengan komputer saat berperan sebagai *webserver*. Komputer mampu menangani sekitar 19405 permintaan dalam setiap detiknya, sedangkan mesin virtual bisa menangani sekitar 18089 permintaan. Hasil perbandingan tersebut juga dapat dilihat grafik perbandingan pada Gambar 4.45.

**Tabel 4.20:** Hasil pengujian mesin virtual dan komputer sebagai *web-server* dengan Apache Benchmark

Perangkat	Permintaan per Detik
Komputer	19405
Mesin Virtual (VM)	18089



**Gambar 4.45:** Kemampuan menerima permintaan setiap detiknya

## DAFTAR PUSTAKA

- [1] M. Armbrust, A. Fox, R. Griffith, et al., Above the cloud : A Berkeley View of Cloud Computing. No. UCB/EECS-2009-28, USA: EECS Department, University of California at Berkeley, February 2009. (Dikutip pada halaman 1).
- [2] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, "Comparison of open-source cloud management platforms: Openstack and opennebula," in Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on, pp. 2457–2461, May 2012. (Dikutip pada halaman 1, 7).
- [3] B. Dordevic, S. Jovanovic, and V. Timcenko, "Cloud computing in amazon and microsoft azure platforms: Performance and service comparison," in Telecommunications Forum Telfor (TELFOR), 2014 22nd, pp. 931–934, Nov 2014. (Dikutip pada halaman 1).
- [4] "Openstack installation guide for ubuntu 14.04." <http://docs.openstack.org/>. Terakhir diakses pada tanggal 18 Mei 2015. (Dikutip pada halaman 5, 7, 16).
- [5] M. Mahjoub, A. Mdhaftar, R. Halima, and M. Jmaiel, "A comparative study of the current cloud computing technologies and offers," in Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on, pp. 131–134, Nov 2011. (Dikutip pada halaman 8).
- [6] W. Zhang, H. Xie, and R. Hsu, "Automatic memory control of multiple virtual machines on a consolidated server," Cloud Computing, IEEE Transactions on, vol. PP, no. 99, pp. 1–1, 2015. (Dikutip pada halaman 9).
- [7] "Phoronix test suite tests." <http://openbenchmarking.org/tests/pts>. Terakhir diakses pada tanggal 31 Mei 2015. (Dikutip pada halaman 54, 62, 69, 72).
- [8] J. Layton, "Finding memory bottlenecks with stream." <http://www.admin-magazine.com/HPC/Articles/>



Finding-Memory-Bottlenecks-with-Stream. Terakhir diakses pada tanggal 31 Mei 2015. (Dikutip pada halaman 57).

- [9] K. Raman, "Optimizing memory bandwidth on stream triad." <https://software.intel.com/en-us/articles/optimizing-memory-bandwidth-on-stream-triad>. Terakhir diakses pada tanggal 31 Mei 2015. (Dikutip pada halaman 57).
- [10] "Cloud servers benchmarks." <http://www.sherweb.com/blog/>. Terakhir diakses pada tanggal 31 Mei 2015. (Dikutip pada halaman 59, 63).

## BAB 5

### PENUTUP

#### 5.1 Kesimpulan

Dari hasil implementasi dan pengujian mesin virtual pada komputasi awan yang sudah dilakukan dapat ditarik beberapa kesimpulan sebagai berikut :

1. Di perlukannya penggunaan jaringan eksternal pada setiap *compute node*, hal tersebut diperlukan oleh nova-compute dalam melakukan pembagian jaringan dan alamat IP bagi setiap mesin virtual. Pada pengujian jaringan, jaringan eksternal digabung menjadi satu dengan jaringan manajemen. Hasilnya mesin virtual dapat dibuat, tetapi tidak berjalan dan tidak memperoleh alokasi alamat IP.
2. Hasil pengujian batas maksimal mesin virtual, jumlah mesin virtual yang dapat dijalankan bergantung pada jumlah memori (RAM) komputer induk. Semakin besar memori dari komputer induk maka semakin besar jumlah mesin virtual yang bisa dibuat, total dari memori yang dapat digunakan oleh pengguna memiliki batas perbandingan 1.5 : 1 dari memori pada komputer induk.
3. Setiap peningkatan 1 core pada CPU dapat berpengaruh pada berkurangnya waktu pemrosesan beban sebesar 18 s. Sedangkan pada *bandwidth* memori atau kecepatan membaca dan menulis data pada memori oleh *processor*, diperoleh rata-rata peningkatan kecepatan mesin virtual sebesar 444,5 MB/s untuk proses *copy*, 415,93 MB/s untuk *scale*, 450,39 MB/s untuk *triad* dan 442,35 MB/s untuk *add*. Selain itu waktu *encoding* juga berkurang sebesar 1,64 s, artinya proses tersebut semakin cepat.
4. Setiap penambahan 1 GB memori mampu mempengaruhi *throughput* rata-rata mesin virtual sebesar 1205,83 MB/s untuk pemrosesan data *integer* dan 1208,01 MB/s untuk pemrosesan *floating-point*, dimana rata-rata tersebut diperoleh melalui proses *add*, *copy*, *scale* dan *triad*. Sedangkan pada *cache* me-

mori mesin virtual, bandwidth justru berkurang sebesar 1,9 MB/s untuk proses baca, bertambah sebesar 38,02 MB/s saat proses tulis dan bertambah lagi sebesar 93,4 MB/s saat proses *read/write/modify* setiap penambahannya.

## 5.2 Saran

Demi pengembangan lebih lanjut mengenai tugas akhir ini, disarankan beberapa langkah lanjutan sebagai berikut :

1. Penambahan alamat IP publik supaya komputasi awan dapat diakses dari luar dan penelitian dapat dilakukan lebih luas.
2. Penambahan *Networking node* supaya pengaturan jaringan beserta sistem keamanan pada komputasi awan lebih tertata dan memiliki perangkat pengatur jaringan pada komputasi awan yang lebih spesifik.
3. Dilakukan penambahan memori atau komputer server baru agar dapat membuat lebih banyak mesin virtual untuk menangani kebutuhan komputasi yang beragam di Lab B201.



**FINAL PROJECT - TE 1599**

## **VIRTUAL MACHINE PERFORMANCE MEASUREMENT ON CLOUD COMPUTING**

Nur Rohman Widiyanto  
NRP 2209100025

Advisor  
Mochamad Hariadi, ST., M.Sc., Ph.D.  
Christyowidiasmoro, ST., MT

Departement of Electrical Engineering  
Faculty of Industrial Technology  
Sepuluh Nopember Institute of Technology  
Surabaya 2015



# ABSTRAK

Nama Mahasiswa : Nur Rohman Widiyanto  
Judul Tugas Akhir : Pengukuran Performansi Mesin Virtual pada Komputasi Awan  
Pembimbing : 1. Mochamad Hariadi, ST., M.Sc., Ph.D.  
2. Christyowidiasmoro, ST., MT.

Berawal kebutuhan komputasi yang semakin beragam saat ini, maka dibangunlah komputasi awan untuk memenuhi kebutuhan tersebut. Komputasi awan dengan salah satu jenis layanannya yaitu IaaS (*Infrastructure as a Service*) memiliki keuntungan tidak perlu membeli komputer fisik, dan konfigurasi mesin virtual yang dapat dirubah dengan mudah. Di lakukan juga pengukuran performa dari mesin virtual yang merupakan layanan dari komputasi awan dengan spesifikasi yang beragam, selain itu performa dari mesin virtual juga dibandingkan dengan sebuah komputer dengan spesifikasi yang sama. Hal tersebut dilakukan guna memperoleh referensi berupa perbandingan performa dari mesin virtual dengan spesifikasi yang beragam, selain itu diperolehnya perbandingan performa antara mesin virtual yang merupakan layanan dari komputasi awan dengan sebuah komputer dengan spesifikasi yang sama. Komputasi awan dibangun menggunakan OpenStack sebagai sistem operasi utamanya. Pengguna juga dapat dengan mudah memilih sistem operasi yang dibutuhkan, mengatur spesifikasi sesuai dengan kebutuhan dan menjalankannya melalui antarmuka *website*. Pengukuran performa mesin virtual menggunakan Phoronix, sebuah aplikasi *open source* untuk *benchmark*. Phoronix memiliki sekitar 130 tes, seperti C-Ray yang digunakan untuk mengukur beban kerja pada *processor*. Hasil pengukuran tersebut menunjukkan bahwa performa dari mesin virtual tidak kalah dengan performa komputer. Setiap peningkatan 1 *core* pada CPU dapat berpengaruh pada berkurangnya waktu pemrosesan beban sebesar 18 s yang diukur dengan C-Ray.

Kata Kunci : IaaS (*Infrastructure as a Service*), Komputasi Awan, Mesin Virtual, OpenStack, Performa

# ABSTRACT

Name : Nur Rohman Widiyanto  
Title : *Virtual Machine Performance Measurement on Cloud Computing*  
Advisors : 1. Mochamad Hariadi, ST., M.Sc., Ph.D.  
2. Christyowidiasmoro, ST., MT.

*Beginning with diverse need of computation, the cloud computing was built to fulfill those needs. Cloud computing with one of its service type named IaaS (Infrastructure as a Service) has the advantage of not needing a physical computer, and easily changeable configuration of virtual machine. performance measuring is also done in the form of cloud computing with various specifications, moreover virtual machine's performance is compared with a computer having similar specifications. It is done in order to obtain references, in the form of performance comparison between virtual machine, which is a cloud computing service, with the Computer of similar specifications. Cloud computing is built with OpenStack as its main Operating System. Users can also easily choose the operating system needed, configure the needed specifications, and run it through the website interface. Virtual Machine's performance is measured using Phoronix, an open source application for benchmark. Phoronix owns 130 tests, like C-Ray which can be used to measure the workload in the processor. The measurement result show that the virtual machine performance is on par with Computer. Every 1 core in CPU affects in the reduced processing time amounting to 18 s which is measured using C-Ray.*

*Keywords : Cloud computing, IaaS (Infrastructure as a Service), Virtual machine, OpenStack, performance*

# KATA PENGANTAR

Puji dan syukur kehadiran Allah SWT atas segala limpahan berkah, rahmat, serta hidayah-Nya, penulis dapat menyelesaikan penelitian ini dengan judul **Analisa Performansi Mesin Virtual pada Komputasi Awan**.

Penelitian ini disusun dalam rangka pemenuhan bidang riset di Jurusan Teknik Elektro ITS, Bidang Studi Teknik Komputer dan Telematika, serta digunakan sebagai persyaratan menyelesaikan pendidikan S1. Penelitian ini dapat terselesaikan tidak lepas dari bantuan berbagai pihak. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Keluarga, Ibu, Bapak dan Saudara tercinta yang telah memberikan dorongan spiritual dan material dalam penyelesaian buku penelitian ini.
2. Bapak Dr. Tri Arief Sardjono, ST., MT. selaku Ketua Jurusan Teknik Elektro, Fakultas Teknologi Industri, Institut Teknologi Sepuluh Nopember.
3. Secara khusus penulis mengucapkan terima kasih yang sebesar-besarnya kepada Bapak Mochamad Hariadi, ST., M.Sc., Ph.D. dan Bapak Christyowidiasmoro, ST., MT. atas bimbingan selama mengerjakan penelitian.
4. Bapak-ibu dosen pengajar Bidang Studi Teknik Komputer dan Telematika, atas pengajaran, bimbingan, serta perhatian yang diberikan kepada penulis selama ini.
5. Seluruh teman-teman *B201-crew* Laboratorium Bidang Studi Teknik Komputer dan Telematika.

Kesempurnaan hanya milik Allah SWT, untuk itu penulis memohon segenap kritik dan saran yang membangun. Semoga penelitian ini dapat memberikan manfaat bagi kita semua. Amin.

Surabaya, Juni 2015

Penulis



# DAFTAR ISI

<b>Abstrak</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>KATA PENGANTAR</b>	<b>v</b>
<b>DAFTAR ISI</b>	<b>vii</b>
<b>DAFTAR GAMBAR</b>	<b>ix</b>
<b>DAFTAR TABEL</b>	<b>xi</b>
<b>DAFTAR KODE</b>	<b>xiii</b>
<b>1 PENDAHULUAN</b>	<b>1</b>
1.1 Latar belakang . . . . .	1
1.2 Permasalahan . . . . .	2
1.3 Tujuan . . . . .	2
1.4 Batasan masalah . . . . .	2
1.5 Sistematika Penulisan . . . . .	2
1.6 Relevansi . . . . .	3
<b>2 TINJAUAN PUSTAKA</b>	<b>5</b>
2.1 OpenStack . . . . .	5
2.2 Virtualisasi . . . . .	7
2.3 Phoronix Test Suite . . . . .	8
<b>3 DESAIN DAN IMPLEMENTASI SISTEM</b>	<b>11</b>
3.1 Desain Sistem . . . . .	11
3.2 Desain Jaringan . . . . .	12
3.3 Alur Implementasi Sistem . . . . .	14
3.4 Perangkat Keras dan Sistem Operasi . . . . .	14
3.5 Pembuatan Controller Node . . . . .	16
3.5.1 Keystone (Identity Service) . . . . .	17
3.5.2 Glance (Image Service) . . . . .	19
3.5.3 Nova (Compute) pada Controller . . . . .	21



3.5.4	Cinder (Block Storage Service) . . . . .	23
3.6	Pembuatan Compute Node . . . . .	25
3.7	Pembuatan Horizon (Dashboard) . . . . .	27
<b>4</b>	<b>PENGUJIAN DAN ANALISA</b>	<b>29</b>
4.1	Pengujian Layanan . . . . .	29
4.1.1	Uji Coba Keystone (Identity Service) . . . . .	29
4.1.2	Uji Coba Glance (Image Service) . . . . .	34
4.1.3	Uji Coba Nova (Compute Service) . . . . .	37
4.1.4	Uji Coba Mesin Virtual (Instance) . . . . .	38
4.1.5	Uji Coba Cider (Block Storage Service) . . . . .	46
4.2	Pengujian Topologi Jaringan . . . . .	50
4.3	Kemampuan Maksimal Komputasi Awan . . . . .	50
4.4	Perbandingan Performa Mesin Virtual . . . . .	51
4.4.1	Performa Komputasi (CPU) . . . . .	53
4.4.2	Performa Penyimpanan . . . . .	62
4.4.3	Performa Memori (RAM) . . . . .	65
4.4.4	Apache Benchmark . . . . .	72
<b>5</b>	<b>PENUTUP</b>	<b>75</b>
5.1	Kesimpulan . . . . .	75
5.2	Saran . . . . .	76
	<b>DAFTAR PUSTAKA</b>	<b>77</b>
	<b>Biografi Penulis</b>	<b>79</b>

## DAFTAR TABEL

3.1	Spesifikasi dari <i>controller</i> dan <i>compute node</i> . . . . .	15
3.2	Spesifikasi dari <i>storage node</i> . . . . .	15
4.1	Jumlah spesifikasi dari empat <i>compute node</i> . . . . .	51
4.2	Spesifikasi (flavor) dari mesin virtual . . . . .	52
4.3	Spesifikasi dari komputer yang akan dibandingkan . . . . .	52
4.4	<i>Flavor</i> mesin virtual untuk mengukur performa komputasi. . . . .	53
4.5	<i>Flavor</i> mesin virtual untuk mengukur performa memori. . . . .	53
4.6	Waktu pemrosesan beban kerja antara mesin virtual dan komputer . . . . .	54
4.7	Waktu pemrosesan beban kerja pada mesin virtual . . . . .	56
4.8	Hasil pengujian memori <i>bandwidth</i> pada komputer dan mesin virtual . . . . .	57
4.9	Hasil pengujian performa memori <i>bandwidth</i> pada mesin virtual . . . . .	58
4.10	Hasil pengujian kecepatan proses encoding . . . . .	60
4.11	Waktu yang diperlukan saat <i>encoding</i> . . . . .	61
4.12	Hasil pengujian penyimpanan dengan Aio-Stress . . . . .	63
4.13	Hasil pengujian penyimpanan dengan FIO . . . . .	64
4.14	Hasil pengujian <i>throughput</i> memori saat mengolah data <i>integer</i> pada mesin virtual dan komputer . . . . .	66
4.15	Hasil pengujian <i>throughput</i> memori saat mengolah <i>floating-point</i> pada mesin virtual dan komputer . . . . .	66
4.16	Hasil pengujian <i>throughput</i> saat memproses data <i>integer</i> pada mesin virtual . . . . .	67
4.17	Hasil pengujian <i>throughput</i> saat memproses <i>floating-point</i> pada mesin virtual . . . . .	67
4.18	Hasil perbandingan kecepatan <i>cache</i> memori antara mesin virtual dengan komputer . . . . .	70
4.19	Hasil perbandingan <i>bandwidth cache</i> memori pada mesin virtual . . . . .	71
4.20	Hasil pengujian mesin virtual dan komputer sebagai <i>webserver</i> dengan Apache Benchmark . . . . .	72

# DAFTAR GAMBAR

2.1	Konsep arsitektur dasar OpenStack . . . . .	5
2.2	Arsitektur inti OpenStack. . . . .	6
2.3	Metode virtualisasi . . . . .	7
3.1	Rancangan sistem layanan yang digunakan . . . . .	12
3.2	Desain jaringan . . . . .	13
3.3	Alur implementasi sistem . . . . .	14
3.4	Hubungan antar layanan pada OpenStack . . . . .	16
3.5	Skema interaksi antara pengguna dengan keystone . . . . .	19
4.1	<i>Login screen</i> . . . . .	30
4.2	Penambahan pengguna melalui antarmuka web . . . . .	31
4.3	Daftar pengguna dan layanan melalui <i>console</i> . . . . .	31
4.4	Daftar pengguna dan layanan melalui antarmuka . . . . .	32
4.5	Daftar <i>tenant</i> melalui <i>console</i> . . . . .	33
4.6	Pengguna dan layanan saat pembuatan <i>project</i> . . . . .	33
4.7	Daftar pengguna dan <i>role</i> melalui <i>console</i> . . . . .	34
4.8	Penambahan <i>image</i> melalui antarmuka . . . . .	35
4.9	Daftar <i>image</i> yang berhasil terunggah dilihat melalui console . . . . .	36
4.10	Daftar layanan dari nova yang sedang berjalan dilihat melalui console . . . . .	36
4.11	Daftar layanan dari nova yang sedang berjalan, bila dilihat melalui antarmuka <i>web</i> . . . . .	37
4.12	Daftar <i>image</i> sistem operasi dari glance yang terbaca oleh nova . . . . .	37
4.13	Daftar <i>key pairs</i> melalui <i>console</i> . . . . .	38
4.14	Pembuatan <i>keypair</i> melalui antarmuka . . . . .	39
4.15	Daftar <i>flavor</i> melalui <i>console</i> . . . . .	39
4.16	Daftar <i>image</i> melalui antarmuka web . . . . .	40
4.17	Daftar <i>nova-network</i> melalui <i>console</i> . . . . .	40
4.18	Daftar secgroup melalui <i>console</i> . . . . .	41
4.19	Pembuatan mesin virtual melalui antarmuka . . . . .	42
4.20	Penambahan <i>flavor</i> melalui antarmuka . . . . .	42
4.21	Tampilan monitoring <i>hypervisors</i> . . . . .	43



4.22	Perintah tampilkan alamat VNC yang berupa URL .	43
4.23	Akses mesin virtual melalui <i>browser</i> . . . . .	44
4.24	Pengaturan <i>security group</i> melalui antarmuka . . . .	45
4.25	Penambahan <i>security group</i> melalui antarmuka . . . .	46
4.26	Melihat daftar mesin melalui <i>console</i> . . . . .	47
4.27	Layanan-layanan pembangun cinder . . . . .	47
4.28	Penambahan <i>volume</i> melalui antarmuka . . . . .	48
4.29	Penambahan penyimpanan melalui antarmuka . . . .	49
4.30	Desain jaringan tanpa jaringan eksternal . . . . .	50
4.31	Perbedaan waktu pemrosesan beban kerja antara mesin virtual dengan komputer . . . . .	54
4.32	Perbedaan waktu pemrosesan beban kerja pada mesin virtual . . . . .	56
4.33	Perbandingan memori bandwidth antara mesin virtual dengan komputer . . . . .	58
4.34	Perbedaan memori bandwidth pada mesin virtual . .	59
4.35	Perbandingan kecepatan encoding . . . . .	60
4.36	Perbedaan waktu saat encoding . . . . .	61
4.37	Perbandingan bandwidth penyimpanan . . . . .	63
4.38	Perbandingan I/O penyimpanan dengan FIO . . . .	64
4.39	Perbandingan <i>throughput</i> saat memproses <i>integer</i> pada komputer dan mesin virtual . . . . .	66
4.40	Perbandingan <i>throughput</i> saat memproses <i>floating point</i> antara mesin virtual dengan komputer . . . . .	68
4.41	Perbedaan <i>throughput</i> saat memproses data <i>integer</i> pada mesin virtual. . . . .	68
4.42	Perbedaan <i>throughput</i> saat memproses <i>floating-point</i> pada mesin virtual . . . . .	69
4.43	Perbandingan kecepatan <i>cache</i> memori antara mesin virtual dengan komputer . . . . .	70
4.44	Perbedaan <i>bandwidth cache</i> memori mesin virtual . .	71
4.45	Kemampuan menerima permintaan setiap detik nya .	73



## DAFTAR KODE

3.1	Galat saat pemasangan glance . . . . .	20
3.2	Hubungan antara database, keystone dan glance . . .	20
3.3	Hubungan database, keystone, glance dan nova . . .	22
3.4	Hubungan antara database, keystone, dan cinder . . .	24
3.5	Hubungan nova pada <i>compute node</i> dengan <i>controller node</i> . . . . .	26
3.6	Nova-network pada <i>controller node</i> . . . . .	27
3.7	Nova-network pada <i>compute node</i> . . . . .	27
3.8	Penghubung <i>controller node</i> dengan antarmuka <i>web</i> . . .	28
4.1	Login script pada keystone dengan nama pengguna admin. . . . .	30
4.2	Perintah untuk tambah pengguna . . . . .	31
4.3	Perintah untuk tambah <i>tenant</i> atau <i>project</i> . . . . .	32
4.4	Perintah untuk memberi <i>role</i> ke sebuah <i>project</i> . . . .	33
4.5	Perintah untuk mengunggah <i>image</i> ke glance . . . . .	34
4.6	Perintah untuk menambahkan <i>keypair</i> ke nova . . . .	38
4.7	Perintah untuk membuat mesin virtual ( <i>instance</i> ) melalui <i>console</i> . . . . .	41
4.8	Perintah untuk memberikan izin agar mesin virtual dapat diakses melalui SSH . . . . .	45
4.9	Perintah untuk memberikan izin agar mesin virtual dapat menerima paket ICMP . . . . .	45
4.10	Perintah untuk menambah <i>volume</i> melalui <i>console</i> . .	48
4.11	Perintah untuk menambah <i>volume</i> ke mesin virtual melalui <i>console</i> . . . . .	49

## BIOGRAFI PENULIS



Nur Rohman Widiyanto, lahir pada 26 Maret 1991 di Lamongan, Jawa Timur. Penulis lulus dari SMP Negeri 1 Paciran pada tahun 2006 kemudian melanjutkan pendidikan ke SMA Muhammadiyah 1 Gresik hingga akhirnya lulus pada tahun 2009. Penulis kemudian melanjutkan pendidikan Strata satu ke Jurusan Teknik Elektro ITS Surabaya bidang studi Teknik Komputer dan Telematika. Saat di kuliah penulis aktif menjadi staff PSDM (Pemberdayaan Sumber Daya Mahasiswa) BEM ITS 2010/2011. Penulis juga aktif menjadi Asisten laboratorium B201 (Telematika) hingga saat ini dan pernah menjabat sebagai koordinator asisten Lab B201 periode 2012/2013. Selama masa kuliah penulis aktif dalam mengikuti ajang perlombaan seperti PKM (Program Kreativitas Mahasiswa), aktif dalam *development group networking* dan juga sebagai *administrator* jaringan Lab B201. Penulis sangat tertarik dengan segala hal yang berhubungan dengan komputer, dan berencana mendalami cabang ilmu komputer lain selain jaringan komputer.

# BAB 1

## PENDAHULUAN

Penelitian ini di latar belakang oleh berbagai kondisi yang menjadi acuan. Selain itu juga terdapat beberapa permasalahan yang akan dijawab sebagai luaran dari penelitian.

### 1.1 Latar belakang

Kebutuhan akan komputasi yang semakin meningkat, seperti pada level pengguna atau *server*. Pada komputasi awan (*cloud computing*) pengguna dapat memperoleh berbagai layanan yang berupa komputasi (CPU), memori, penyimpanan data dan lain sebagainya. Semuanya disediakan sebagai sebuah layanan oleh pihak ketiga [1]. Jenis dari layanan itu adalah IaaS (*Infrastructure as a Service*) yang memiliki keuntungan diantaranya pengguna tidak perlu membeli komputer fisik, melainkan hanya cukup melakukan konfigurasi pada mesin virtual dengan mudah [2]. Konfigurasi mesin virtual dapat dirubah dengan mudah, contohnya saat komputer virtual tersebut mengalami kelebihan beban maka dapat ditambahkan CPU, RAM, media penyimpanan dan sebagainya dengan segera.

Keuntungan lainnya adalah pengurangan biaya investasi perangkat keras, kemudahan *backup* serta *recovery*, pengurangan pemanfaatan data center, pengurangan biaya sewa slot server, kemudahan skalabilitas dan pengelolaan [3]. Di sisi lain muncul beberapa pertanyaan bagaimana performa sistem operasi yang dipasang pada komputer atau mesin virtual, dan bagaimana perbandingan performa dari mesin virtual tersebut dengan sistem operasi yang dipasang sebuah komputer dengan spesifikasi yang sama.

OpenStack berperan sebagai sistem operasi pada komputasi awan, fitur yang dimiliki mendukung pembuatan dan penyediaan IaaS. Bentuk layanan yang dibuat dalam penelitian ini berupa mesin virtual yang kemudian diukur performanya. Selain itu juga diukur performa dari sebuah komputer, keduanya menggunakan sistem operasi dan spesifikasi yang sama. Hal ini dilakukan guna memperoleh hasil perbandingan performa antara mesin virtual dengan sebuah komputer dan juga mengukur performa mesin virtual dengan



spesifikasi yang terus dinaikan demi memperoleh referensi dalam pemilihan kebutuhan komputasi yang tepat.

## **1.2 Permasalahan**

Berawal dari permasalahan akan kebutuhan komputasi yang beragam dalam skala lab, maka dibangunlah komputasi awan untuk memenuhi kebutuhan tersebut. Layanan yang dibangun berupa IaaS (*Infrastructure as a Service*) dengan mesin virtual sebagai layanan utamanya. Di lakukan juga pengukuran performa dari mesin virtual yang merupakan layanan dari komputasi awan dengan spesifikasi yang beragam, selain itu performa dari mesin virtual juga dibandingkan dengan sebuah komputer menggunakan spesifikasi yang sama.

## **1.3 Tujuan**

Tujuan utama dari penelitian ini adalah membangun komputasi awan skala lab demi memenuhi kebutuhan komputasi yang beragam di dalamnya. Di perolehnya referensi berupa perbandingan performa dari mesin virtual dengan spesifikasi yang beragam, selain itu di dapat juga perbandingan performa antara mesin virtual yang merupakan layanan dari komputasi awan dengan sebuah komputer dengan spesifikasi yang sama. Manfaat lain dari penelitian ini adalah diperolehnya referensi untuk mengatasi peningkatan kebutuhan komputasi yang beragam dengan cara penggunaan yang mudah.

## **1.4 Batasan masalah**

Batasan masalah yang timbul dari permasalahan Tugas Akhir ini adalah :

1. Layanan komputasi awan yang dibangun berupa IaaS (*Infrastructure as a Service*) dengan menyediakan mesin virtual skala lab.

## **1.5 Sistematika Penulisan**

Laporan penelitian Tugas akhir ini tersusun dalam sistematika dan terstruktur sehingga mudah dipahami dan dipelajari oleh pembaca maupun seseorang yang ingin melanjutkan penelitian ini. Alur sistematika penulisan laporan penelitian ini yaitu :



## 1. BAB I Pendahuluan

Bab ini berisi uraian tentang latar belakang permasalahan, penegasan dan alasan pemilihan judul, sistematika laporan, tujuan dan metodologi penelitian.

## 2. BAB II Dasar Teori

Pada bab ini berisi tentang uraian secara sistematis teori-teori yang berhubungan dengan permasalahan yang dibahas pada penelitian ini. Teori-teori ini digunakan sebagai dasar dalam penelitian, yaitu informasi terkait teknologi komputasi awan, mesin virtual, dan teori-teori penunjang lainnya.

## 3. BAB III Perancangan Sistem dan Impementasi

Bab ini berisi tentang penjelasan-penjelasan terkait sistem yang akan dibuat. Guna mendukung itu digunakanlah blok diagram atau *work flow* agar sistem yang akan dibuat dapat terlihat dan mudah dibaca untuk implentasi pada pelaksanaan tugas akhir.

## 4. BAB IV Pengujian dan Analisa

Bab ini menjelaskan tentang pengujian yang dilakukan terhadap sistem dalam penelitian ini dan menganalisa sistem. Spesifikasi perangkat keras dan perangkat lunak yang diuji juga disebutkan dalam bab ini. Sehingga ketika akan dikembangkan lebih jauh, spesifikasi perlengkapannya bisa dipenuhi dengan mudah tanpa harus melakukan ujicoba perangkat lunak maupun perangkat keras lagi.

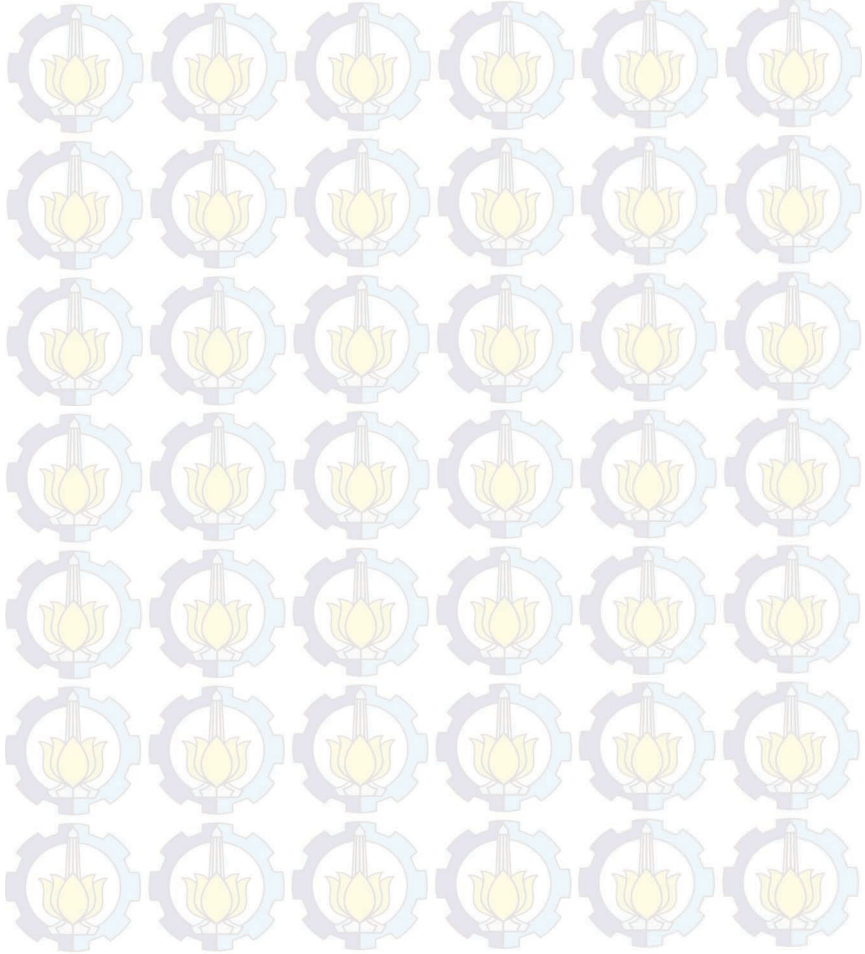
## 5. BAB V Penutup

Bab ini merupakan penutup yang berisi kesimpulan yang diambil dari penelitian dan pengujian yang telah dilakukan. Saran dan kritik yang membangun untuk mengembangkan lebih lanjut juga dituliskan pada bab ini.

# 1.6 Relevansi

Penelitian mengenai komputasi awan merupakan bidang penelitian yang sangat dibutuhkan dan dipakai dalam pemenuhan kebutuhan komputasi yang semakin beragam saat ini. Layanan *Infras-structure as a Service* (IaaS) dengan mesin virtual sebagai layanan utamanya. Di mana pengguna dapat mengatur kebutuhannya se-

perti memori, penyimpanan dan komputasi (CPU) dengan mudah dan cepat. Dari penelitian ini dihasilkan sebuah hasil pengukuran antara mesin virtual layanan dari komputasi awan dan sebuah komputer dengan spesifikasi yang sama. Tujuannya diperoleh referensi bagi pengguna dalam pemilihan komputasi yang tepat.



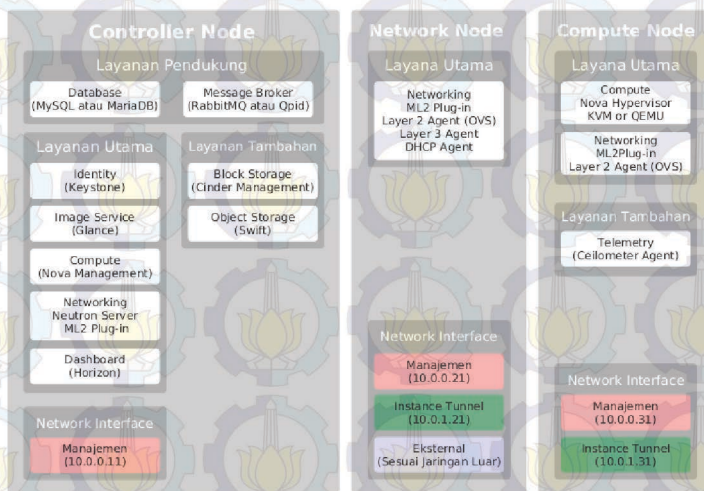
## BAB 2

### TINJAUAN PUSTAKA

Demi mendukung penelitian ini, dibutuhkan beberapa teori penunjang sebagai bahan acuan dan refrensi. Dengan demikian penelitian ini menjadi lebih terarah.

#### 2.1 OpenStack

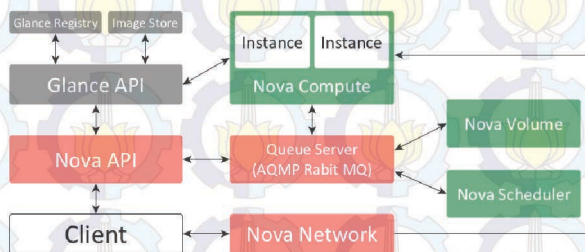
OpenStack adalah sistem operasi pada komputasi awan yang mengontrol *big pool*, penyimpanan data, dan jaringan di seluruh data center, semua dikelola melalui *dashboard* yang memberikan kontrol *administrator* yang ditampilkan dan diakses melalui *browser* dengan skema seperti pada Gambar 2.1.



Gambar 2.1: Konsep arsitektur dasar OpenStack

1. OpenStack Networking atau Neutron adalah sebuah sistem untuk mengatur jaringan pada OpenStack, didalamnya terdapat API yang berfungsi untuk mengelola jaringan dan alamat IP (*internet protocol*) bagi penggunaannya [4].

2. Swift atau Object Storage merupakan media penyimpanan obyek pada OpenStack. Swift dilengkapi dengan *proxy server*, *object server*, sebuah *account server*, sebuah *container server* dan *ring*. Sebuah penyimpanan jangka panjang dengan data yang statis, dapat diambil dan diperbaharui. Fungsi utama dan fiturnya adalah sebagai media penyimpanan yang besar dan aman, mengurangi redudansi data, kemampuan arsip dan media *streaming*.
3. Nova atau OpenStack *Compute* merupakan bagian utama dari OpenStack, bertugas sebagai kontroler dari sistem komputasi awan. Memiliki enam komponen seperti Nova-API, *Message Queue* dengan Rabbitmq, Nova-Compute, Nova-Network, Nova-Volume dan Nova-Scheduler. Seluruh komponen pada arsitektur Nova mengikuti aturan *shared-nothing* dan *messaging-based*, maksud dari *shared-nothing* adalah setiap komponen dapat dipasang pada server manapun. Misalnya, *compute controller*, *volume controller*, *network controller* dan *object storage* dapat dipasang dalam satu server atau empat server secara terpisah. Seperti pada Gambar 2.2. *Queue Server* berada ditengah-tengah arsitekstur, maksud dari messaging-based adalah terjalin komunikasi pada setiap kontroler pada komputasi awan diantaranya *volume*, *network*, dan penjadwalan melalui *queue server* pada *advanced message queue protocol* (AMQP).



Gambar 2.2: Arsitektur inti OpenStack.

4. OpenStack *Image Service* dibuat untuk mencari dan mengambil *Image* dari mesin virtual. Fitur ini dinamakan Glance,

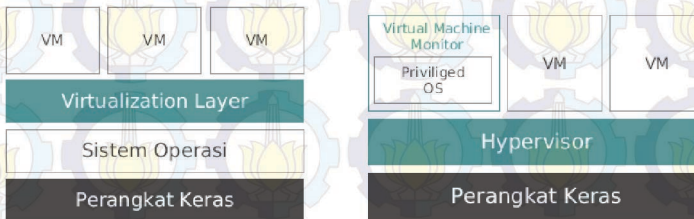


*Glance Registry* dan *Glance Control* merupakan bagian utamanya.

5. Keystone atau Identify Service adalah keamanan utama pada komputasi awan yang menyediakan layanan authentication dan authorization dari OpenStack [2].
6. Cinder atau *Block Storage* menyediakan blok penyimpanan atau volume dalam menjalankan layanan OpenStack. Dengan arsitektur yang mudah diatur setiap penambahan blok penyimpanan datanya.
7. Dashboard atau Horizon disediakan untuk administrator dan pengguna yang berupa tampilan antar muka untuk mengakses, mengatur secara langsung melalui browser pada layanan komputasi awan [4].

## 2.2 Virtualisasi

Virtualisasi (*Virtualization*) adalah cara yang memungkinkan sebuah komputer atau mesin fisik untuk menjalankan beberapa komputer secara virtual di atasnya dengan menggunakan batuan perangkat lunak.



(a) *Full-virtualization*

(b) *Para-virtualization*



(c) *Isolation*

**Gambar 2.3:** Metode virtualisasi

Tiga metode virtualisasi diantaranya *full-virtualization*, *para-virtualization* dan *OS-Level virtualization* (Isolasi). *Full-virtualization* menyediakan mesin virtual mirip atau menyesuaikan dengan spesifikasi perangkat keras pada komputer induk seperti pada Gambar 2.3 (a). Tipe ini menawarkan keamanan, kemudahan pemasangan dan pemindahan pada setiap mesin virtual. Contoh dari pengguna metode virtualisasi ini adalah produk-produk dari VMware, VirtualBox dan KVM.

Pada Gambar 2.3 (b), *para-virtualization* menyediakan mesin virtual yang lebih abstrak. Mesin virtual tersebut memiliki spesifikasi yang mirip tetapi tidak persis dengan perangkat keras dari komputer induknya. Xen dan HyperV menggunakan metode virtualisasi jenis ini.

Pembagian sumber daya antar pengguna mesin virtual yang diatur oleh *virtual machine manager* seperti pada Gambar 2.3 (c). Sejumlah mesin virtual berjalan diatas beberapa salinan kernel sistem operasi yang sama dengan komputer induk. Melalui proses tersebutlah maka terbentuk sebuah mesin virtual melalui sebuah kernel sistem operasi. Metode ini hanya terdapat di linux, contohnya OpenVZ.

*Isolasion* dinilai memiliki performa paling baik. namun kelemahannya metode tersebut hanya bisa digunakan pada Linux dan sitem operasi yang dilakukan harus sama dengan sistem operasi dari komputer induk. Sedangkan *para-virtualisasi* dibuat dengan memodifikasi kernel agar mampu melakukan virtualisasi. *Full-virtualization* hanya dapat dibangun dengan perangkat keras dan processor yang mendukung virtualisasi yang akan dimodifikasi kernelnya sebelum melakukan virtualisasi. Namun dalam mengakses perangkat keras dari komputer induk, mesin virtul harus melewati layer virtualisasi terlebih dahulu seperti pada Gambar 2.3 (a) [5].

## 2.3 Phoronix Test Suite

Phoronix Test Suite (versi 5.6.0; PTS) [28] adalah perangkat lunak open-source yang digunakan untuk melakukan pengujian *benchmark*, selain itu PTS juga dapat berjalan pada banyak sistem operasi (*multiplatform*) seperti Linux, Mac OSX, BSD dan Microsoft Windows. *Benchmark* sendiri adalah suatu metode untuk meli-

hat performa atau kemampuan komputer melauai tes, dari hasil tes tersebut kemudian dibandingkan dengan komputer lainnya. *Benchmarking* biasanya akan menghasilkan hasil akhir berupa angka atau skor. Dengan membandingkan skor dari hasil proses tersebut, maka akan terlihat komputer dengan performa yang lebih baik.

Lebih dari 130 tes profil dan 60 rangkaian pengujian pada Phoronix Test Suite dapat digunakan. Pengguna harus melakukan download setiap rangkaian atau profilnya saat pertama kali melakukan *benchmarking*, tetapi untuk pemakaian selanjutnya tidak perlu melakukan download. Mulai dari kemampuan komputasi (CPU), memori, proses penyimpanan (I/O) dan pengolahan grafis pada komputer, perangkat bergerak dan komputasi awan. Pengguna tidak perlu menjalankan 130 tes profil untuk melakukan pengujian, cukup memilih beberapa rangkain atau tes yang dibutuhkan [6].



## BAB 3

# DESAIN DAN IMPLEMENTASI SISTEM

Penelitian ini dilaksanakan sesuai dengan desain sistem berikut dengan implementasinya. Desain sistem merupakan konsep dari pembuatan dan perancangan infrastruktur dan kemudian diwujudkan dalam bentuk blok-blok alur yang harus dikerjakan. Pada bagian implementasi merupakan pelaksanaan teknis untuk setiap blok pada desain sistem.

### 3.1 Desain Sistem

Penelitian ini bertujuan untuk mengukur performa mesin virtual yang merupakan layanan dari komputasi awan dan sebuah komputer. Spesifikasi dan sistem operasi yang sama bagi keduanya adalah sebuah bagian yang wajib. Sebelum melakukan pengukuran performa, infrastruktur dan sistem dari komputasi awan harus dibangun terlebih dahulu.

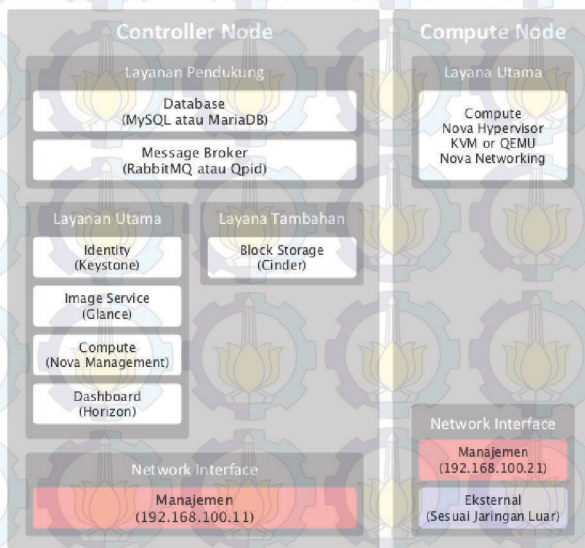
Pembangun infrastruktur komputasi awan, perangkat lunak yang digunakan adalah OpenStack. Fungsi utamanya adalah untuk melakukan kontrol dan menyediakan layanan mesin virtual. Bagian terpenting dari OpenStack adalah *controller* dan *compute node*. *Controller node* menjalankan keystone (*identity service*), glance (*image service*), nova-controller sebagai pengontrol nova-compute beserta nova-network pada setiap *compute node*, cinder (*block storage service*), dan *dashboard* untuk tampilan yang dapat diakses melalui *browser*. Selain itu beberapa layanan pendukung pada *controller node* diatarannya MariaDB untuk *database*, *message broker* untuk komunikasi antar layanan dengan menggunakan RabbitMQ, dan NTP (*Network Time Protocol*) yang bertugas untuk sinkronisasi waktu antar komputer.

Pada *compute node* berfungsi untuk menjalankan hypervisors, di sana merupakan tempat mesin virtual dioperasikan. Jumlah dari *compute node* dapat terus ditambah sesuai kebutuhan mesin virtual yang ingin dipakai. Sedangkan untuk Neuron (*networking node*)



berperan untuk menjalankan layanan pada layer 2 TCP/IP, pengaturan jaringan luar, sebagian layer 3 seperti NAT (*Network Address Translation*) dan DHCP (*Dynamic Host Configuration Protocol*). Namun *networking node* tidak digunakan dalam penelitian ini, karena tersedianya *gateway* yang mengatur jaringan.

Rancangan dasar dari OpenStack seperti pada Gambar 3.1 terdiri dari dua *node* dan terdapat berbagai layanan penyusun dan layanan tambahan di dalamnya. Layanan tambahan seperti Swift (*Object Storage*), Trove (*Database service*), Heat (*Orchestration*), dan ceilometer (*Telemetry*) tidak digunakan dalam penelitian ini karena masih belum diperlukan. Sedangkan layanan tambahan yang digunakan adalah cinder (*Block storage*). Jumlah dari *controller node* dan *compute node* bisa terus ditambah sesuai kebutuhan.

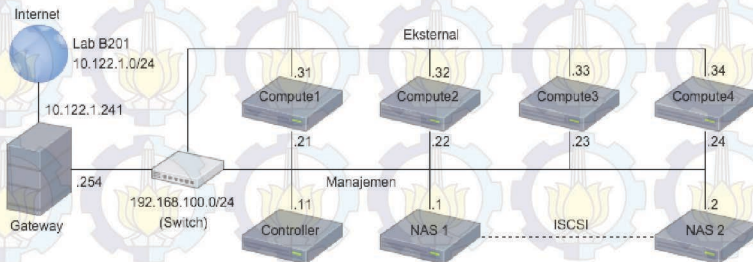


Gambar 3.1: Rancangan sistem layanan yang digunakan

### 3.2 Desain Jaringan

Desain jaringan pada penelitian ini menggunakan jaringan privat pada laboratorium Telematika (B201). Alamat IP (*Internet*

*Protocol*) dari *gateway* yang digunakan untuk mengakses jaringan luar adalah 10.122.1.241, berada dalam jaringan 10.122.1.0/24 milik laboratorium Telematika (B201). Sedangkan IP alias yang digunakan untuk implementasi komputasi awan adalah 192.168.100.0/24 dengan alamat IP *gateway* 192.168.100.254 seperti pada Gambar 3.2.



**Gambar 3.2:** Desain jaringan

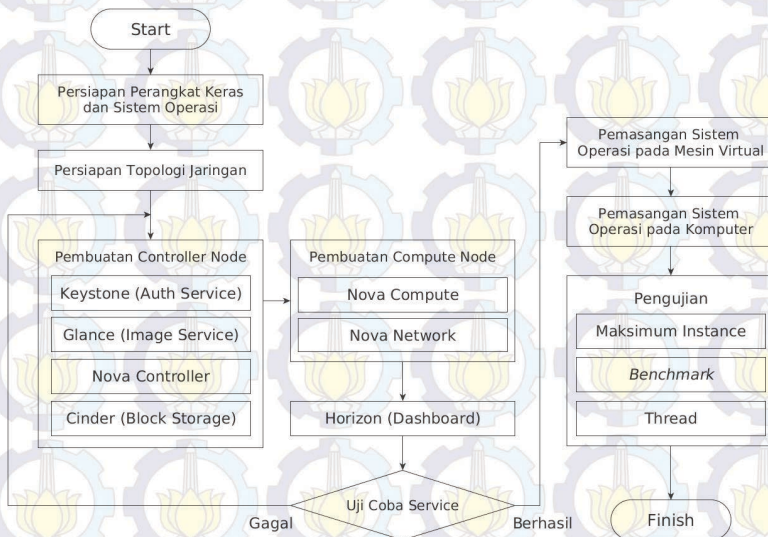
Seperti yang terlihat pada Gambar 3.2, tiga unit komputer yang terhubung dengan *gateway* melalui *switch*. Di antaranya satu *controller node* dan dua *compute node*. Controller terhubung dengan jaringan manajemen, begitu juga compute1, compute2, compute3 dan compute4. Fungsi utama dari jaringan manajemen adalah untuk sinkronisasi dan komunikasi antar *node*, Controller berperan sebagai pengatur keseluruhan sistem komputasi awan. Kemudian compute1, compute2, compute3 dan compute4 terhubung dengan jaringan eksternal. Hal ini bertujuan untuk memberikan alokasi alamat IP pada mesin virtual yang merupakan layanan dari sistem komputasi awan.

Jaringan manajemen dan eksternal pada umumnya dibuat berberda jaringan, misalkan untuk jaringan manajemen hanya menggunakan IP privat atau lokal sedangkan jaringan eksternal serharusnya menggunakan IP publik atau IP yang sengaja dibuat untuk diakses oleh banyak pengguna. Pada penelitian ini alamat IP sengaja dibuat sama, karena gateway dari komputasi awan sudah terdaftar dalam routing table gateway laboratorium Telematika. Hal ini bertujuan untuk membatasi pengguna komputasi awan hanya sebatas laboratorium Telematika.

Terdapat dua *storage node* dengan nama NAS1 dan NAS2, keduanya saling terhubung melalui jaringan manajemen. Tugas utama dari keduanya adalah sebagai NAS (*Network-attached Storage*) yang menyediakan penyimpanan dan terhubung dengan cinder melalui jaringan manajemen. Media penyimpanan pada NAS2 digabungkan menjadi satu melalui jaringan dengan NAS1 menggunakan ISCSI (*Internet Small Computer System Interface*).

### 3.3 Alur Implementasi Sistem

Alur kerja dalam pengerjaan tugas akhir ini terbagi mejadi tujuh tahapan proses dan satu percabangan. Dalam masing-masing proses memiliki luaran yang dihasilkan dan menjadi input dari proses selanjutnya seperti yang terlihat pada Gambar 3.3.



Gambar 3.3: Alur implementasi sistem

### 3.4 Perangkat Keras dan Sistem Operasi

Komputer yang dipakai dalam penelitian ini sejumlah tujuh unit, satu komputer berperan sebagai controller node dan empat komputer lainnya sebagai compute node. Komputer yang digunak-



an sebagai *controller* dan *compute* memiliki spesifikasi yang sama seperti pada Tabel 3.1. Sedangkan untuk dua unit lainnya merupakan NAS (*Network Attached Storage*) dengan spesifikasi yang sama namun berbeda pada jumlah Disk Storage seperti pada Tabel 3.2.

**Tabel 3.1:** Spesifikasi dari *controller* dan *compute node*

Spesifikasi	Keterangan
Jumlah Prosesor	2
Jumlah Core	Quad-core
Kecepatan Prosesor	3.46 GHz
Jenis Prosesor	Intel® Xeon® 5600 series
<i>Cache</i>	8 MB L3
Arsitektur	x86_64
Memori	6 GB
<i>Disk Storage</i>	300 GB
<i>Power Supplay</i>	460 Watt

**Tabel 3.2:** Spesifikasi dari *storage node*

Spesifikasi	Keterangan
Jumlah Prosesor	2
Jumlah Core	Quad-core
Kecepatan Prosesor	3.46 GHz
Jenis Prosesor	Intel® Xeon® 5600 series
<i>Cache</i>	8 MB L3
Arsitektur	x86_64
Memori	6 GB
<i>Disk Storage</i>	
- NAS1	6927 GB
- NAS2	7927 GB
<i>Power Supplay</i>	460 Watt

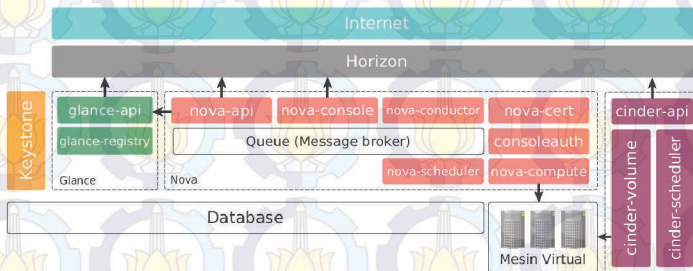


Spesifikasi minimum dari OpenStack berbeda-beda setiap node-nya, untuk *controller node* minimum 1 prosesor, 2 GB memori dan 5 GB *disk storage*. Kemudian pada *compute node* adalah 1 prosesor, 2 GB memori dan 10 GB *disk storage*, sedangkan pada *networking node* adalah 1 prosesor, 2 GB memori dan 5 GB *disk storage*. OpenStack mendukung beberapa sistem operasi linux di antaranya Suse Linux Enterprise Server dan OpenSuse, RedHat, Fedora, CentOS, Ubuntu dan Debian [4].

Sistem operasi yang akan digunakan dalam penelitian ini adalah Ubuntu 14.04 (Trusty Tahr) dengan arsitektur 64-bit pada *controller node* dan *compute node*. Versi dari OpenStack yang digunakan adalah Juno yang rilis pada bulan Oktober 2014. Sedangkan pada *storage node* sistem operasi yang digunakan adalah Windows Server Storage 2008 R2 dengan Arsitektur 64-bit.

### 3.5 Pembuatan Controller Node

Pemasangan perangkat lunak pada komputer yang akan berperan sebagai *controller node*, terdiri dari beberapa layanan yang saling terhubung diantaranya MariaDB untuk manajemen *database*, rabbitmq, glance, keystone, nova dan horizon.



Gambar 3.4: Hubungan antar layanan pada OpenStack

Beberapa layanan seperti keystone, glance dan nova terbentuk dari beberapa dependence atau perangkat lunak yang menjadi penyusun terbentuknya layanan. Sehingga setiap layanan mampu berjalan dan terhubung seperti pada Gambar 3.4. Tidak hanya menunjukkan hubungan antar layanan, melainkan juga menunjuk-

an susunan layanan pembangun yang saling ketergantungan antara satu layanan dengan layanan lainnya. Penjelasan dari pembuatan layanan dari OpenStack akan dijelaskan pada bagian selanjutnya. Berikut merupakan penjelasan dua bagian penting yang mendasari berjalannya setiap layanan OpenStack :

1. RabbitMQ

Messaging server atau message broker dapat menggunakan RabbitMQ yang bertugas untuk mengkoordinasi informasi dan komunikasi antar server dalam OpenStack.

2. Database

Database digunakan sebagai penyimpanan *metadata* dari setiap layanan OpenStack. Setiap pembuatan layanan dari OpenStack tentunya harus terdaftar terlebih dahulu ke *database*.

### 3.5.1 Keystone (Identity Service)

Keystone atau *identity service* tugas utamanya terbagi dua di antaranya :

1. *User Managent* : Identifikasi *user* dan *permission*.
2. *Service Catalog* : Menyediakan katalog dari *service* yang tersedia dengan menggunakan API (*application programming interface*) *endpoint* dari OpenStack.

Bagian-bagian penting yang harus dimengerti dalam penggunaan keystone seperti dijabarkan pada beberapa point berikut :

1. *User* merupakan representasi dari pengguna layanan, sistem, atau layanan dari OpenStack yang terhubung dengan layanan komputasi awan. Keystone mengatur validasi setiap layanan yang diminta oleh pengguna. Pengguna diharuskan untuk *login*, kemudian memperoleh *token* untuk mengakses layanan tersebut.
2. *Credential* merupakan data pengguna seperti nama pengguna (*username*) dan *password*, nama pengguna dan API *key*, atau sebuah *token authentication* yang disediakan oleh keystone.
3. *Authentication* adalah sebuah konfirmasi untuk mengetahui identitas untuk pengguna yang sedang mengirim permintaan untuk menggunakan dari layanan OpenStack. Data-data kon-

firmasi yang dimasukan oleh pengguna tersebut adalah *credential*.

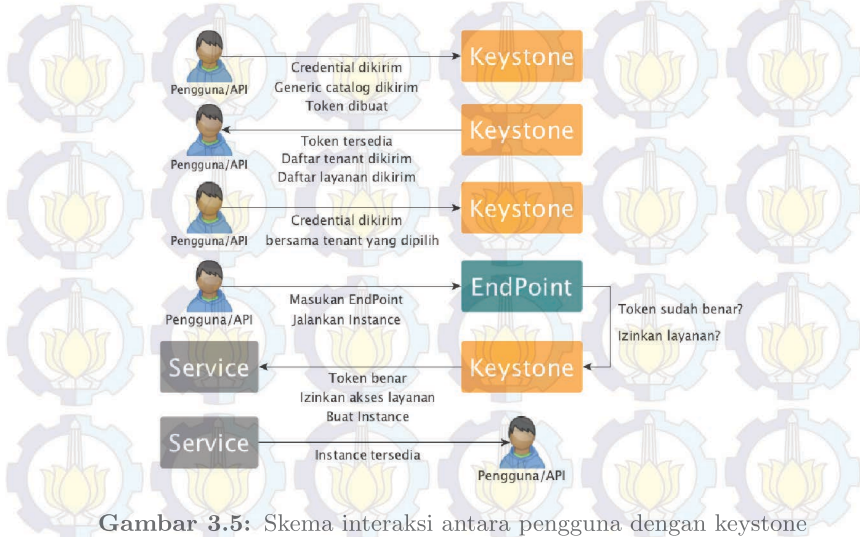
4. *Token* adalah sebuah teks acak yang digunakan untuk mengakses layanan dari OpenStack. Masing-masing *token* mendeskripsikan setiap layanan OpenStack dan *token* akan dihapus dalam jangka waktu tertentu.
5. *Tenant* merupakan sebuah wadah yang digunakan untuk mengelompokkan layanan dari OpenStack yang terhubung dengan keystone. Penelompokan berdasarkan akun, layanan, group atau yang lainnya sesuai dengan kebutuhan.
6. *Service* merupakan istilah yang mewakili layanan dari OpenStack seperti glance, cinder dan nova. Layanan-layanan tersebut mampu menyediakan satu atau lebih *endpoint* yang nantinya bisa diakses oleh pengguna.
7. *Endpoint* adalah sebuah alamat yang dapat diakses oleh pengguna, biasanya berupa URL (*Uniform Resource Locator*).
8. *Role* adalah sebuah aturan yang didefinisikan oleh pengguna untuk melakukan suatu operasi atau perintah tertentu.
9. Keystone *client* adalah sebuah layanan *command line* pada OpenStack. Seperti perintah keystone service-create dan keystone endpoint-create yang digunakan untuk registrasi layanan.

Interaksi antara pengguna dengan keystone yang berperan sebagai pengatur perizinan (otorisasi) dan pengguna (*user*) ditunjukkan pada Gambar 3.5. Di mana pengguna mengirimkan autentikasi ke OpenStack untuk dengan memasukan nama pengguna dan *password*, data inilah yang disebut dengan *credential*. Pengguna yang membuat sebuah *token* secara acak dengan openssl untuk dikirim kepada keystone. Ketika *credential* yang dimasukan oleh pengguna benar, maka layanan dari OpenStack akan berjalan. Sebelum layanan OpenStack berjalan, keystone akan memberikan *list tenant* yang berupa layanan-layanan dari OpenStack untuk dipilih oleh pengguna dan dikirim lagi ke keystone besersamaan dengan *token*.

Kemudian pengguna memilih layanan yang ingin diakses dengan memasukan *endpoint*, bentuk dari *enpoint* adalah berupa URL.



*Endpoint* akan diperiksa kembali tokennya dan layanan apa saja yang ingin diakses oleh pengguna melalui keystone. Setelah itu keystone akan memberi perintah ke layanan tujuan untuk menjalankan *instance* (mesin virtual), akhirnya pengguna dapat mengakses *instance* yang berupa layanan utama dari OpenStack yang berupa mesin virtual tersebut.



Gambar 3.5: Skema interaksi antara pengguna dengan keystone

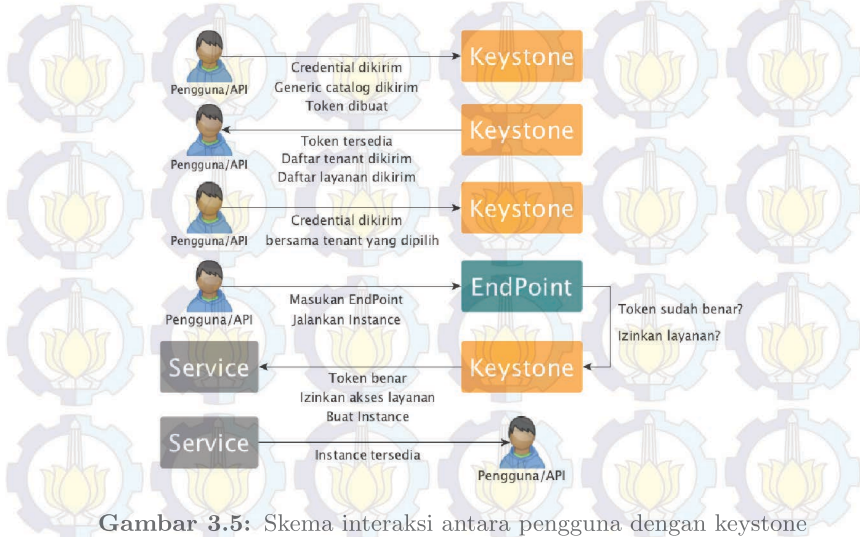
### 3.5.2 Glance (Image Service)

Glance bertugas untuk pendaftaran, pencarian dan pengambilan image dari mesin virtual. Setiap *metadata image* diurutkan atau biasa disebut dengan proses *query* agar memudahkan pencarian *image*. Glance bukan tempat penyimpanan *image*, biasanya disimpan pada *object storage* atau tempat yang lain. Berikut merupakan bagian-bagian penting dari glance :

1. **Glance-api** bertugas menerima perintah simpan, cari dan ambil *image*.
2. **Glance-registry** tugas utamanya adalah melakukan penyimpanan, pemrosesan dan pengambilan metadata seperti tipe dan ukuran *image*.
3. **Database** untuk penyimpanan *metadata* dari *image*.



*Endpoint* akan diperiksa kembali tokennya dan layanan apa saja yang ingin diakses oleh pengguna melalui keystone. Setelah itu keystone akan memberi perintah ke layanan tujuan untuk menjalankan *instance* (mesin virtual), akhirnya pengguna dapat mengakses *instance* yang berupa layanan utama dari OpenStack yang berupa mesin virtual tersebut.



Gambar 3.5: Skema interaksi antara pengguna dengan keystone

### 3.5.2 Glance (Image Service)

Glance bertugas untuk pendaftaran, pencarian dan pengambilan image dari mesin virtual. Setiap *metadata image* diurutkan atau biasa disebut dengan proses *query* agar memudahkan pencarian *image*. Glance bukan tempat penyimpanan *image*, biasanya disimpan pada *object storage* atau tempat yang lain. Berikut merupakan bagian-bagian penting dari glance :

1. **Glance-api** bertugas menerima perintah simpan, cari dan ambil *image*.
2. **Glance-registry** tugas utamanya adalah melakukan penyimpanan, pemrosesan dan pengambilan metadata seperti tipe dan ukuran *image*.
3. **Database** untuk penyimpanan *metadata* dari *image*.

Pada proses pemasangan *dependence* dari glance terjadi galat di akhir proses, tepatnya saat melakukan perintah `apt-get`. Akibatnya adalah beberapa skrip python pembangun OpenStack gagal tereksekusi, seperti yang ditunjukkan pada Kode 3.1. Pemecahan dari masalah ini adalah penambahan perintah `LC_ALL=en_US.UTF-8` dan `LANG=en_US.UTF-8` pada *environment* sistem operasi.

```
File "/usr/lib/python2.7/dist-packages
/keystone/openstack/common/gettextutils.py",
line 207, in translate_msgid system_locale =
locale.getdefaultlocale()
File "/usr/lib/python2.7/locale.py", line 543, in
getdefaultlocale return _parse_localename (localename)
File "/usr/lib/python2.7/locale.py", line 475,
in _parse_localename raise
ValueError, 'unknown locale: %s' %localename
ValueError: unknown locale: UTF-8
dpkg: error processing package keystone (--configure):
subprocess installed post-installation
script returned error exit status 1
Processing triggers for libc-bin (2.19-0ubuntu6.1)
Processing triggers for ureadahead (0.100.0-16)
Errors were encountered while processing:
glance
E: Sub-process /usr/bin/dpkg returned an error
code (1)
```

### Kode 3.1: Galat saat pemasangan glance

Hal Pertama yang harus dilakukan dalam proses pemasangan glance adalah pembuatan *database*. Kemudian dilanjutkan dengan pendaftarkan glance sebagai layanan pada keystone, tentukan *role*, *tenant* dan *endpoint*. Berkas (*file*) penting yang menghubungkan *database*, keystone dan glance terdapat dalam `/etc/glance/glance-api.conf` seperti yang dituliskan pada Kode 3.2.

```
[DEFAULT]
...
verbose = True

[database]
...
connection = mysql://glance:GLANCE_DBPASS@controller/
glance
```

```
[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = glance
admin_password = GLANCE_PASS

[paste_deploy]
flavor = keystone
```

**Kode 3.2:** Hubungan antara database, keystone dan glance

### 3.5.3 Nova (Compute) pada Controller

Nova atau *compute service* merupakan bagian utama dari IaaS (*Infrastructure as a Service*) sistem yang bertugas sebagai *host* dan sistem manajemen pada komputasi awan. Nova yang terhubung langsung dengan keystone untuk autentikasi, glance untuk *image* dari sistem operasi dan *dashboard* merupakan tempat untuk interaksi antara pengguna dengan layanan dan konfigurasinya. Bagian utama dari nova terdiri dari tiga bagian di antaranya API, *Compute Core*, dan *Networking VMs*. Setiap bagian dari nova terdiri dari beberapa perangkat lunak dasar demi berjalannya nova. Nova yang terpasang pada *controller node* bertugas sebagai pengontrol dari nova yang terpasang pada setiap *compute node*. Berikut merupakan penjelasan setiap bagian perangkat lunak pendukung nova pada *controller node* :

#### API (Application Programming interface)

Pada Gambar 3.4 terlihat API dari nova terhubung dengan glance untuk mengakses dan berkoordinasi perihal *image*. Di sini terjalin hubungan antara horizon (*dashboard*) dengan nova agar layanan OpenStack dapat diakses melalui *browser*. API dari nova terletak pada *controller node*. Berikut merupakan penjelasan perangkat lunak pendukung API pada nova :

1. **Nova-api** berfungsi untuk menerima dan merespon panggilan dari pengguna compute API.
2. **Nova-api-metadata** bertugas menerima permintaan meta-



data dari instances. Nova-api-metadata biasanya digunakan dalam penggunaan compute node dalam jumlah banyak.

## Compute Core

Tugas utama dari *compute core* adalah menyediakan layanan komputasi atau *instance* yang berupa mesin virtual. Jumlah dari *compute core* dapat terus ditambah secara horizontal sesuai kebutuhan komputasi. Berikut ini adalah perangkat lunak pendukung *compute core* :

1. **nova-scheduler** adalah penerima permintaan instance atau mesin virtual dari queue, kemudian menentukan compute node mana yang akan menjalankannya.
2. **nova-conductor** merupakan sebuah perangkat lunak penghubung antara nova-compute dan database.

Sama seperti pada glance, hal pertama yang dilakukan dalam pemasangan nova adalah pemasangan layanan pendukung dan dilanjutkan pembuatan *database*. Kemudian dilanjutkan dengan mendaftarkan nova sebagai layanan pada keystone, tentukan juga *role*, *tenant* dan *endpoint*. File penting yang menghubungkan *database*, keystone, glance dan nova terdapat pada `/etc/nova/nova.conf` seperti yang dituliskan dalam Kode 3.3.

```
[DEFAULT]
...
verbose = True
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
my_ip = 192.168.100.11
vncserver_listen = 192.168.100.11
vncserver_proxyclient_address = 192.168.100.11
network_api_class = nova.network.api.API
security_group_api = nova
compute_driver = libvirt.LibvirtDriver

[database]
...
connection = mysql://nova:NOVADB_PASS@controller/nova

[glance]
...
```



```
host = controller

[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = nova
admin_password = NOVA_PASS
```

**Kode 3.3:** Hubungan database, keystone, galance dan nova

### 3.5.4 Cinder (Block Storage Service)

Cinder bertugas untuk penyediaan penyimpanan tambahan untuk mesin virtual. Pada *block storage service* hanya terdapat satu layanan, yaitu cinder atau *block storage service*. Dalam layanan tersebut terdapat dua layanan pembangun, agar cinder dapat berjalan dan memberikan layanan berupa media penyimpanan. Kedua layanan pembangun tersebut adalah cinder-api, cinder-scheduller dan cinder-volume. Berikut merupakan penjelasan dari ketiga layanan pembangun tersebut :

1. **cinder-api** menerima perintah dan mengarahkannya ke cinder-volume untuk proses eksekusi.
2. **cinder-volume** berinteraksi langsung dengan cinder, dan proses seperti cinder-scheduler. Proses tersebut dilakukan melalui *mesage queue*. Cinder-volume merespon untuk membaca dan menulis permintaan yang kemudian dikirim ke cinder untuk membuat penyimpanan sesuai dengan permintaan pengguna. Cinder-volume dapat berinteraksi dengan berbagai macam media penyimpanan dan berbagai macam driver.
3. **cinder-scheduler** menerima perintah dan mengarahkannya ke cinder-volume untuk proses eksekusi.

Pertama kali yang dilakukan pada proses pemasangan cinder pada *controller node* adalah pemasangan layanan pembangun yang terdiri dari cinder-api, cinder-volume, cinder-scheduler dan dilanjutkan pembuatan *database* untuk cinder. Kemudian dilanjutkan dengan pendaftarkan cinder sebagai layanan pada keystone, tentuk-

an juga *role*, *tenant* dan *endpoint*. *File* penting yang menghubungkan *database*, *keystone* dan *cinder* terdapat pada `/etc/cinder/cinder.conf` seperti yang dituliskan dalam Kode 3.4.

```
[DEFAULT]
...
verbose = True
debug = True
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
auth_strategy = keystone
my_ip = 192.168.100.11
glance_host = controller
volume_driver = cinder.volume.drivers.nfs.NfsDriver
nfs_mount_options = rsize=8192,wsize=8192,timeo=14,intr
nfs_mount_point_base = /var/lib/cinder/mnt
nfs_oversub_ratio = 1.0
nfs_shares_config = /etc/cinder/nfs_share
nfs_sparsed_volumes = False
nfs_used_ratio = 0.95

[database]
...
connection = mysql://cinder:CINDERDB_PASS@controller/
cinder

[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = cinder
admin_password = CINDER_PASS
```

**Kode 3.4:** Hubungan antara *database*, *keystone*, dan *cinder*

Pada Kode 3.4 selain adanya hubungan antara *database*, *keystone* dan *cinder*. Terdapat juga hubungan antara *controller* dengan *storage node*, dimana *controller* terpasang *cinder* sedangkan *storage node* yang terdiri dari *NAS1* dan *NAS2* yang menyediakan layanan *NFS (Network File System)*.

Hubungan anantara *storage node* dan *controller node* pada `nfs_shares_config = /etc/cinder/nfs_share`, dimana contro-

ler dapat melakukan *mount* (penambahan penyimpanan) secara otomatis dengan membaca isi *file* yang terletak pada direktori `/etc/cinder/nfs.share`. Isi dari *file* tersebut adalah alamat server dari NFS (`192.168.100.1:/Cinder`). Akses ke NFS server dilakukan melalui alamat `192.168.100.1` milik NAS1. Hal tersebut dikarenakan penggunaan ISCSI yang menggabungkan penyimpanan milik NAS2 ke NAS1 melalui jaringan manajemen.

Pada `volume_driver = cinder.volume.drivers.nfs.NfsDriver` menunjukkan driver penyimpanan yang digunakan oleh cinder, seperti pada pembahasan sebelumnya bahwa jenis penyimpanan yang digunakan adalah NFS. Kemudian pada `nfs_mount_point_base = /var/lib/cinder/mnt` dapat diartikan bahwa pada direktori `/var/lib/cinder/mnt` adalah tempat penyimpanan yang ditambahkan oleh NFS. Selanjutnya pada `nfs_mount_options = rsize=8192, wsize=8192, timeo=14, intr` konfigurasi yang menentukan *permission* dari direktori yang ditambahkan oleh NFS, apakah *read*, *write* dan *execute*.

### 3.6 Pembuatan Compute Node

Pada *compute node* hanya terdapat satu layanan, yaitu nova atau *compute service*. Dalam layanan tersebut terdapat dua layanan pembangun, agar nova dapat berjalan dan layanan mesin virtual (*instance*) dapat digunakan. Kedua perangkat lunak itu adalah nova-compute dan nova-network. Nova-compute berada pada bagian *compute core* dan nova-network dan nova-dhcpbridge pada *networking VMs*, berikut penjelasannya :

1. **nova-compute** bertugas menjalankan proses pembuatan dan penghapusan layanan yang berupa mesin virtual melalui hypervisor API seperti pada Gambar 3.4. Contoh dari hypervisor API adalah XenAPI untuk XenServer/XCP, libvirt untuk KVM atau QEMU, VMware API untuk VMware. Dalam penelitian ini hypervisor API yang dipakai adalah libvirt untuk KVM atau QEMU. Letak dari nova-compute berada pada *compute node*, jumlahnya terus bisa ditambah sesuai kebutuhan komputasi.
2. **nova-network** berkerja mirip seperti nova-compute, dengan menerima permintaan layanan berupa kebutuhan jaringan dan



menjalankannya untuk memenuhi permintaan tersebut. Seperti konfigurasi *bridge* dan perubahan aturan pada *iptables* untuk mengatur setiap *instance*.

3. **nova-dhcpbridge** merupakan sebuah skrip yang berfungsi untuk mendeteksi IP Address yang disimpan dalam database dengan menggunakan *dnsmasq* yang merupakan fitur dari *dhcp-script*.

Pertama kali yang dilakukan dalam pemasangan nova pada *compute node* adalah pemasangan layanan pembangun yaitu *nova-compute*. Selanjutnya tambahkan konfigurasi pada *file* yang menghubungkan seluruh *compute node* dengan *controller node*. File tersebut terdapat pada `/etc/nova/nova.conf` seperti yang tertulis dalam Kode 3.5.

```
[DEFAULT]
...
my_ip = 192.168.100.21
vnc_enabled = True
vncserver_listen = 0.0.0.0
vncserver_proxyclient_address = 192.168.100.21
novncproxy_base_url =
http://controller:6080/vnc_auto.html
verbose = True

[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = nova
admin_password = NOVA_PASS

[libvirt]
...
virt_type = qemu

[glance]
...
host = controller
```

**Kode 3.5:** Hubungan nova pada *compute node* dengan *controller node*



Pembuatan nova-network pada *compute node* diawali dengan pemasangan *dependence*. Kemudian tambahkan konfigurasi pada *file* yang menghubungkan antara nova pada *controller node* dengan nova pada *compute node*. Hal tersebut bertujuan agar *controller node* dapat terhubung dan memberikan perintah ke semua *compute node*. Kedua *file* tersebut terdapat pada `/etc/nova/nova.conf`. Kode 3.6 merupakan konfigurasi yang harus ditambahkan pada *controller node* dan Kode 3.7 adalah konfigurasi yang harus ditambahkan pada setiap *compute node*.

```
[DEFAULT]
...
network_api_class = nova.network.api.API
security_group_api = nova
```

**Kode 3.6:** Nova-network pada *controller node*

```
[DEFAULT]
...
network_api_class = nova.network.api.API
security_group_api = nova
firewall_driver = nova.virt.libvirt.firewall.
    IptablesFirewallDriver
network_manager = nova.network.manager.FlatDHCPManager
network_size = 254
allow_same_net_traffic = False
multi_host = True
send_arp_for_ha = True
share_dhcp_address = True
force_dhcp_release = True
flat_network_bridge = br100
flat_interface = em2
public_interface = em2
```

**Kode 3.7:** Nova-network pada *compute node*

### 3.7 Pembuatan Horizon (Dashboard)

Openstack *dashboard* disebut juga dengan Horizon adalah sebuah antarmuka berbasis *website* yang berfungsi untuk memudahkan pengguna dalam mengakses setiap layanan dari Openstack. Kemudian mempermudah pengguna dalam mengatur sumber daya dari

untuk komputasi yang digunakannya, seperti memori, penyimpanan dan komputasi (CPU). Selain itu horizon juga mengatur interaksi antara *compute* dengan *controller* melalui Openstack API agar dapat diakses dan diatur melalui *browser*. Pada penggunaan dan pengaksesannya, pengguna diharuskan memakai *browser* yang mendukung HTML5, mengizinkan *cookies* dan javascript.

Beberapa aplikasi pendukung untuk membangun horizon diantaranya apache2 untuk *web server*, memcache dan python versi 2.6 atau 2.7 yang mendukung Django. Setelah pemasangan aplikasi pendukung tersebut dilakukan, tambahkan konfigurasi pada *file* yang dapat menjadikan *controller node* dapat diakses melalui antarmuka web. File tersebut terdapat pada `/etc/openstack-dashboard/local_settings.py` seperti pada Kode 3.8.

```
...
OPENSTACK_HOST = "controller"
...
ALLOWED_HOSTS = ['*']
...
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.
MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    }
}
```

**Kode 3.8:** Penghubung *controller node* dengan antarmuka *web*

## BAB 4

# PENGUJIAN DAN ANALISA

Pada bab ini pengujian dibagi menjadi lima tahapan yaitu pengujian layanan (*service*), pengujian jaringan, kemampuan maksimal komputasi awan, perbandingan performa mesin virtual dan perhitungan jumlah *thread*. Sehingga dengan adanya pengujian tersebut, dapat ditarik beberapa kesimpulan dari pelaksanaan tugas akhir ini.

### 4.1 Pengujian Layanan

Pada bagian ini pengujian dilakukan untuk memastikan apakah layanan pada komputasi awan dapat berjalan atau tidak. Layanan yang akan diuji dalam penelitian ini diantaranya keystone, glance, nova dan cinder yang akan dijabarkan pada bagian selanjutnya. Kemudian dilanjutkan dengan pengujian pemasangan sistem operasi pada mesin virtual (*instance*) yang disediakan oleh komputasi awan.

Penggunaan layanan akan dibandingkan keberhasilannya melalui *console* dengan melakukan *remote* secara langsung ke *controller node*, dan penggunaan layanan yang diakses melalui antarmuka berbasis *website*. Antarmuka tersebut merupakan salah satu layanan utama dari OpenStack yang biasanya disebut dengan horizon (*dashboard*).

#### 4.1.1 Uji Coba Keystone (Identity Service)

Terdapat dua pengguna yang dibuat pada proses implementasi yaitu *admin* dan *user*. Selain itu setiap layanan juga didaftarkan sebagai pengguna dengan *tenant* dan *role* yang berbeda. Pengujian pertama kali yang harus dilakukan adalah masuk (*login*) ke dalam sistem manajemen OpenStack, *login* adalah bagian dari layanan yang disediakan oleh keystone.

Pengguna terlebih dahulu diharuskan melakukan *remote* ke *controller node*. Kemudian membuat bash script seperti pada Kode 4.1 dan menerapkannya ke *environment* sistem operasi untuk *login*. Penggunaan perintah `source admin-openstack.sh` untuk admin

atau `source (nama script).sh` untuk pengguna lain dengan mengisi `OS_TENANT_NAME`, `OS_USERNAME`, dan `OS_PASSWORD` yang berbeda sesuai dengan nama dan *password* dari pengguna.

```
export OS_TENANT_NAME = admin
export OS_USERNAME = admin
export OS_PASSWORD = ADMIN_LOGIN_PASS
export OS_AUTH_URL = http://controller:35357/v2.0
```

**Kode 4.1:** Login script pada keystone dengan nama pengguna admin.

Pengujian keystone melalui antarmuka (*user interface*) ditunjukkan pada Gambar 4.1 dimana pengguna dapat memasukkan nama pengguna (*username*) dan *password* yang telah dibuat sebelumnya. Pengaksesan halaman tersebut dilakukan melalui *browser*, pengguna harus terlebih dahulu mengakses alamat `http://192.168.100.11/horizon`, di mana halaman tersebut merupakan alamat dari *controller node*. Proses pembuatan *username* dan *password* akan dibahas juga pada bagian ini juga.



**Gambar 4.1:** *Login screen*

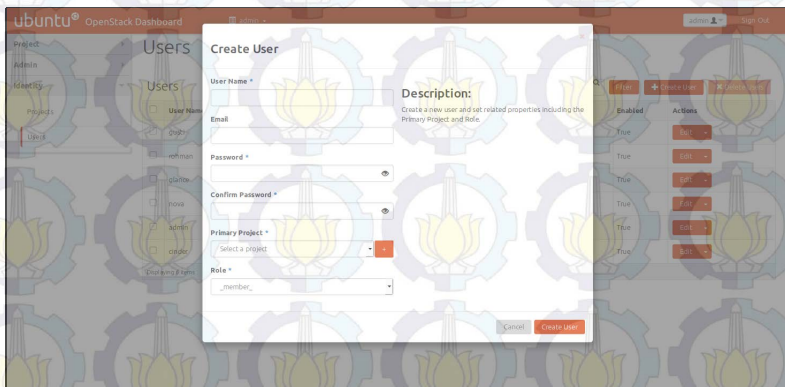
Penggunaan perintah keystone yang lebih spesifik sesuai dengan kebutuhan. Pada Kode 4.2 merupakan perintah untuk penambahan pengguna atau layanan. Sebagai contoh, nama pengguna yang ingin ditambahkan dalam pengujian ini adalah “rohman”. Maka pengguna diharuskan untuk menyesuaikan nama, *password* dan *email* pada skrip tersebut.



```
$ keystone user-create --name rohman --pass ROHMAN_PASS \
--email ROHMAN_EMAIL_ADDRESS
```

#### Kode 4.2: Perintah untuk tambah pengguna

Penambahan pengguna atau layanan melalui antarmuka, *admin* diharuskan memilih menu **Identity** kemudian dilanjutkan dengan **User** dan pilih **Create User** seperti pada Gambar 4.2. Setelah melakukan penambahan pengguna melalui antarmuka, nama pengguna akan tampil pada halaman itu juga. Pada Gambar 4.3 merupakan perintah untuk melihat tabel daftar pengguna, pada gambar tersebut juga dapat dilihat pengguna yang berhasil ditambahkan.



Gambar 4.2: Penambahan pengguna melalui antarmuka web



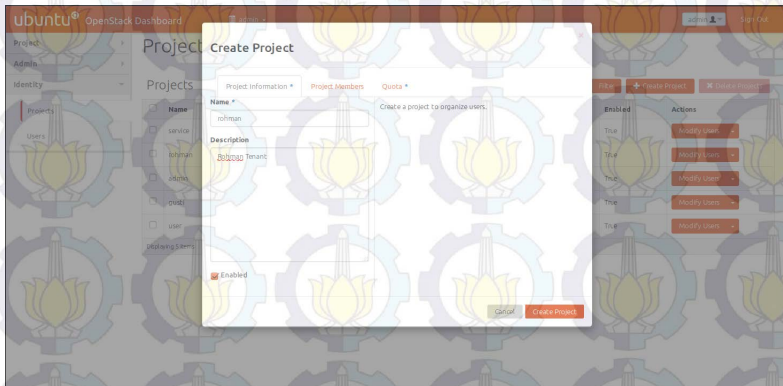
Gambar 4.3: Daftar pengguna dan layanan melalui *console*

Penambahan *tenant* atau *project* dapat dilakukan dengan perintah seperti pada Kode 4.3, dimana dalam perintah tersebut terdapat nama *tenant* dan deskripsinya. Sebagai contoh, nama *tenant* yang ingin ditambahkan dalam pengujian ini adalah “rohman” dengan deskripsi “Rohman Tenant”.

```
$ keystone tenant-create --name rohman --description \
"Rohman Tenant"
```

**Kode 4.3:** Perintah untuk tambah *tenant* atau *project*

Ketika penambahan *tenant* dilakukan melalui antarmuka, yang harus dilakukan oleh *admin* adalah memilih menu **Identity**, kemudian **Project** dan pilih **Create Project**. Kemudian pada **Project Information** isikan nama *tenant* dan deskripsinya seperti pada Gambar 4.4.



**Gambar 4.4:** Daftar pengguna dan layanan melalui antarmuka

Setelah melakukan penambahan *tenant* melalui antarmuka, nama *tenant* dan deskripsinya akan tampil pada tabel daftar di halaman tersebut. Sedangkan pada Gambar 4.5 merupakan perintah untuk melihat daftar *tenant* melalui *console*, pada gambar tersebut juga terlihat *tenant* yang berhasil dibuat. Selanjutnya adalah penambahan *role* ke *tenant* dan pengguna yang dibuat sebelumnya, *role* disini adalah peran dari pengguna. Apakah pengguna tersebut

berperan sebagai *admin* atau *member*. Pada Kode 4.4 merupakan langkah penambahan *role* dengan menggunakan perintah melalui *console*.

```
rohman@controller:~$ keystone tenant-list
```

id	name	enabled
217bf01ccd7747368e022cb9b6a34a75	admin	True
ab86185113d649eb8119bcb4e7db0f5c	gusti	True
18f3ebf7da75485db204b8167efd600f	rohman	True
048ee86912d54a848c461791a6147d0d	service	True
d0159b5f3c4d4127870d7180696040eb	user	True

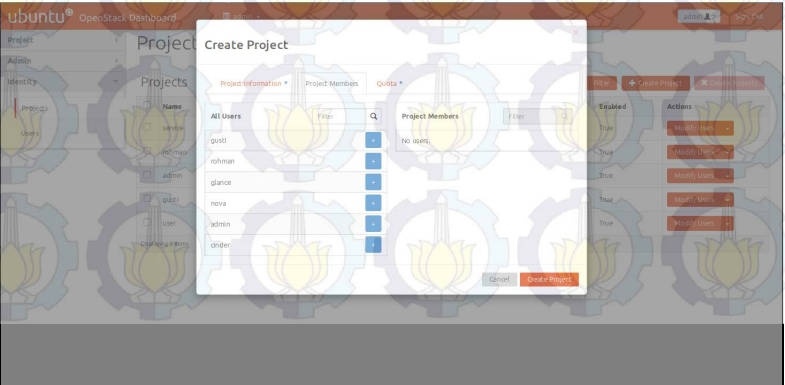
```
rohman@controller:~$
```

Gambar 4.5: Daftar *tenant* melalui *console*

```
$ keystone user-role-add --tenant rohman --user rohman \
--role _member_
```

**Kode 4.4:** Perintah untuk memberi *role* ke sebuah *project*

Pada penambahan *role* ke *tenant* dan pengguna dilakukan melalui antarmuka, yang harus dilakukan oleh *admin* adalah memilih menu **Identity**, kemudian **Project** dan selanjutnya pilih **Create Project**. Pada menu **Create Project** pilih *tab* **Project Member** seperti pada Gambar 4.6. Kemudian pilih pengguna yang akan digunakan. Pada menu *dropdown* pilih *\_member\_* untuk menjadikan pengguna tersebut sebagai *member* atau *admin* untuk mejadikan-nya sebagai *admin*.



Gambar 4.6: Pengguna dan layanan saat pembuatan *project*

Setelah melakukan penambahan *role* ke *tenant* dan pengguna melalui antarmuka, nama *tenant* dan deskripsinya akan tampil pada halaman itu juga. Pada Gambar 4.7 merupakan perintah untuk melihat daftar *role*, yang digunakan untuk mengatur pengaturan hak akses (*permission*) dari pengguna.

```
rohman@controller:~$ keystone role-list
```

id	name
e63bf78394d04225a82fe2a0a4a4170a	_member_
1bb942fddca2423492c81e49e4332590	admin

```
rohman@controller:~$
```

Gambar 4.7: Daftar pengguna dan *role* melalui *console*

Mengacu pada proses yang dijelaskan pada bagian ini, maka dapat disimpulkan bahwa pemasangan keystone sukses. Keystone sudah mampu menyediakan layanan *authentication* mulai dari, proses *login*, pembuatan *user*, *tenant* dan penambahan *role*.

#### 4.1.2 Uji Coba Glance (Image Service)

Glance merupakan layanan penyedia *image* sistem operasi. Pada pengujian ini yang akan dilakukan adalah pengunggahan (*upload*) *file image* sistem operasi ke glance. Pada pengujian ini sistem operasi yang akan diunggah adalah ubuntu-trusty-tahr-14.04-amd64. Seperti pengujian pada keystone dibagian sebelumnya, hal pertama yang harus dilakukan saat melakukan pengujian melalui *console* adalah masuk menggunakan bash skrip yang terlebih dahulu harus dibuat seperti pada Kode 4.1. Selanjutnya gunakan perintah untuk upload image seperti pada Kode 4.5

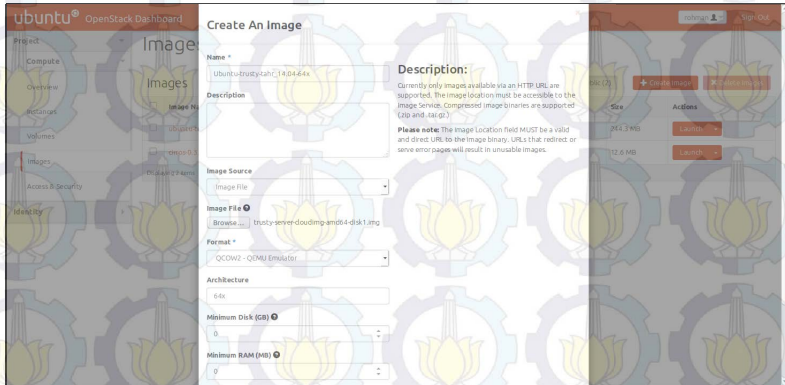
```
$ glance image-create \  
--name "ubuntu-trusty-tahr-14.04-amd64" \  
--file ubuntu-trusty-tahr-14.04-amd64 \  
--disk-format qcow2 --container-format bare \  
--is-public True --progress
```

Kode 4.5: Perintah untuk mengunggah *image* ke glance

Pada Kode 4.5 dapat ditarik kesimpulan bahwa sistem operasi yang diunggah adalah ubuntu versi 14.04 (Trusty Tahr) dengan ti-



pe *disk* *qcow2* yang biasanya digunakan oleh mesin virtual QEMU. Agar dapat diakses oleh seluruh pengguna maka menu *public* diatur sebagai *True*. Untuk melihat apakah *image* sudah berhasil terunggah, gunakan perintah pada Gambar 4.9. Selain itu akan tampil daftar *image* yang pernah diunggah.



Gambar 4.8: Penambahan *image* melalui antarmuka

Pada pengujian glance menggunakan antarmuka, yang harus dilakukan oleh pengguna setelah masuk adalah memilih menu **Project**, kemudian **Compute** dan pilih menu **Image**. Lanjutkan dengan **Create Image** kemudian isi *form* sesuai spesifikasi image, dan unggah *file image* seperti pada Gambar 4.8, kemudian pilih **Create Image**. Setelah berhasil melakukan pengunggahan *Image* melalui antarmuka, nama *Image* dan deskripsinya akan tampil pada tabel *image* halaman itu juga.

Berdasarkan pada setiap langkah pada bagian ini, dimana pengguna berhasil melakukan pengunggahan *image* ke dalam sistem komputasi awan melalui glance. Proses pengunggahan image yang dilakukan melalui console dan antarmuka, keduanya berhasil dilakukan. Maka dapat ditarik kesimpulan bahwa pemasangan glance sukses.

```

rohan@controller:~$ glance image-list
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | Disk Format | Container Format | Size | Status |
+-----+-----+-----+-----+-----+-----+
| 01c66c1-2011-41d2-978a-b6f94e225609 | cirros-0.3.3-x86_64 | qcow2 | bare | 13200896 | active |
| 207a2b4b-46f5-4103-8f58-032da8bc5231 | ubuntu-trusty-fahr-14.04-amd64 | qcow2 | bare | 256180736 | active |
+-----+-----+-----+-----+-----+-----+
rohan@controller:~$ █

```

Gambar 4.9: Daftar *image* yang berhasil terunggah dilihat melalui console

```

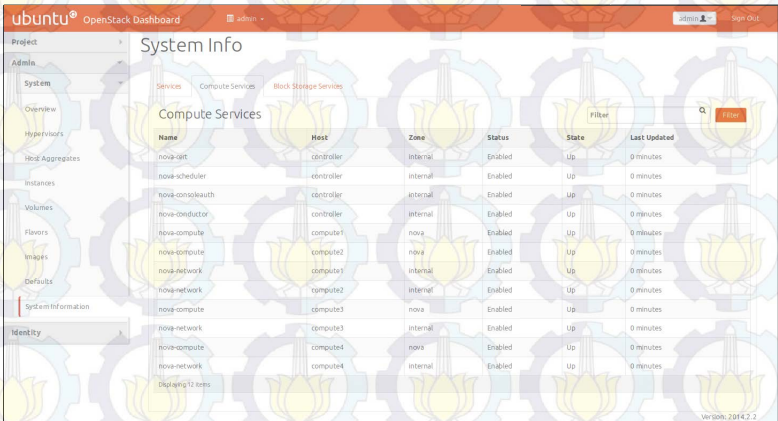
rohan@controller:~$ nova service-list
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Id | Binary | Host | Zone | Status | State | Updated_at | Disabled | Reason |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | nova-cert | controller | Internal | enabled | up | 2015-05-06T06:32:28.000000 | - | - |
| 2 | nova-scheduler | controller | Internal | enabled | up | 2015-05-06T06:32:28.000000 | - | - |
| 3 | nova-consoleauth | controller | Internal | enabled | up | 2015-05-06T06:32:27.000000 | - | - |
| 4 | nova-conductor | controller | Internal | enabled | up | 2015-05-06T06:32:24.000000 | - | - |
| 5 | nova-compute | compute1 | nova | enabled | up | 2015-05-06T06:32:27.000000 | None | - |
| 6 | nova-compute | compute2 | nova | enabled | up | 2015-05-06T06:32:24.000000 | None | - |
| 7 | nova-compute | compute1 | Internal | enabled | up | 2015-05-06T06:32:29.000000 | - | - |
| 8 | nova-network | compute2 | Internal | enabled | up | 2015-05-06T06:32:24.000000 | - | - |
| 9 | nova-compute | compute3 | nova | enabled | up | 2015-05-06T06:32:27.000000 | None | - |
| 10 | nova-network | compute3 | Internal | enabled | up | 2015-05-06T06:32:27.000000 | - | - |
| 11 | nova-compute | compute4 | nova | enabled | up | 2015-05-06T06:32:24.000000 | - | - |
| 12 | nova-network | compute4 | Internal | enabled | up | 2015-05-06T06:32:24.000000 | - | - |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
rohan@controller:~$ █

```

Gambar 4.10: Daftar layanan dari nova yang sedang berjalan dilihat melalui console

### 4.1.3 Uji Coba Nova (Compute Service)

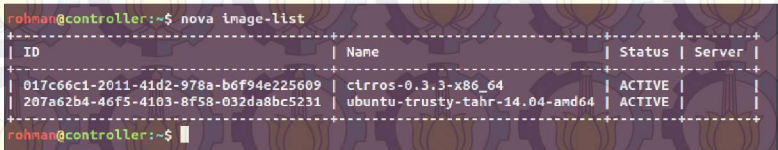
Pengujian pada Nova atau *compute service* dilakukan dengan melakukan pengecekan apakah seluruh layanan yang mendasari nova sudah terpasang dan tidak menunjukkan status galat. Pengecekan dilakukan dengan menggunakan perintah pada *console* seperti pada Gambar 4.10. Sedangkan pengecekan melalui antarmuka dapat dilakukan dengan memilih menu **Admin**, kemudian **System**, dan dilanjutkan dengan memilih **System Information**. Pada *tab Compute Services* akan terlihat seluruh status dari layanan yang membangun nova seperti pada Gambar 4.11. Selain itu terlihat juga nova sukses tersambung dengan glance seperti pada Gambar 4.12, yang mana *image* dari glance terdeteksi oleh nova.



The screenshot shows the OpenStack Dashboard's 'System Info' page. The 'Compute Services' tab is selected, displaying a table of services. The table has columns for Name, Host, Zone, Status, State, and Last Updated. The services listed include nova-api, nova-scheduler, nova-conductor, nova-compute, nova-network, and nova-consoleauth, all of which are in an 'Up' state.

Name	Host	Zone	Status	State	Last Updated
nova-api	controller	internal	Enabled	Up	0 minutes
nova-scheduler	controller	internal	Enabled	Up	0 minutes
nova-conductor	controller	internal	Enabled	Up	0 minutes
nova-compute	compute1	nova	Enabled	Up	0 minutes
nova-compute	compute2	nova	Enabled	Up	0 minutes
nova-network	compute1	internal	Enabled	Up	0 minutes
nova-network	compute2	internal	Enabled	Up	0 minutes
nova-compute	compute3	nova	Enabled	Up	0 minutes
nova-network	compute3	internal	Enabled	Up	0 minutes
nova-compute	compute4	nova	Enabled	Up	0 minutes
nova-network	compute4	internal	Enabled	Up	0 minutes

**Gambar 4.11:** Daftar layanan dari nova yang sedang berjalan, bila dilihat melalui antarmuka *web*



The screenshot shows a terminal window where the command 'nova image-list' has been executed. The output is a table showing two active images: 'clrros-0.3.3-x86\_64' and 'ubuntu-trusty-tahr-14.04-amd64'.

ID	Name	Status	Server
017c66c1-2011-41d2-978a-b6f94e225609	clrros-0.3.3-x86_64	ACTIVE	
207a62b4-46f5-4103-8f58-032da8bc5231	ubuntu-trusty-tahr-14.04-amd64	ACTIVE	

**Gambar 4.12:** Daftar *image* sistem operasi dari glance yang terbaca oleh nova



#### 4.1.4 Uji Coba Mesin Virtual (Instance)

Mesin virtual merupakan layanan utama dari komputasi awan yang dibangun untuk penelitian ini. Pada bagian ini dilakukan pengujian, apakah layanan dari komputasi awan yang berupa mesin virtual bisa digunakan. Langkah awal yang dilakukan adalah penerapan *authentication*, pengguna, *tenant*, dan *endpoint* yang merupakan layanan utama dari keystone seperti pada bagian sebelumnya. Kemudian dilanjutkan dengan glance yang sudah terhubung dengan nova. Uji coba instance dilakukan menggunakan dua cara, yaitu melalui *console* dan antarmuka.

Masuk dengan menuliskan nama pengguna dan password, jika menggunakan antarmuka. Jika menggunakan *console*, maka buat bash skrip untuk masuk seperti pada Kode 4.1. Kemudian tambahkan key pair pada “rohman” untuk mengakses mesin virtual. Buat sebuah *key pair* dengan perintah *ssh-keygen*, selanjutnya tambahkan ke nova dengan perintah seperti Kode 4.6. Gunakan perintah seperti pada Gambar 4.13 untuk melihat apakah *keypair* sudah berhasil ditambahkan.

```
$ nova keypair-add --pub-key ~/.ssh/id_rsa.pub rohman-key
```

Kode 4.6: Perintah untuk menambahkan *keypair* ke nova

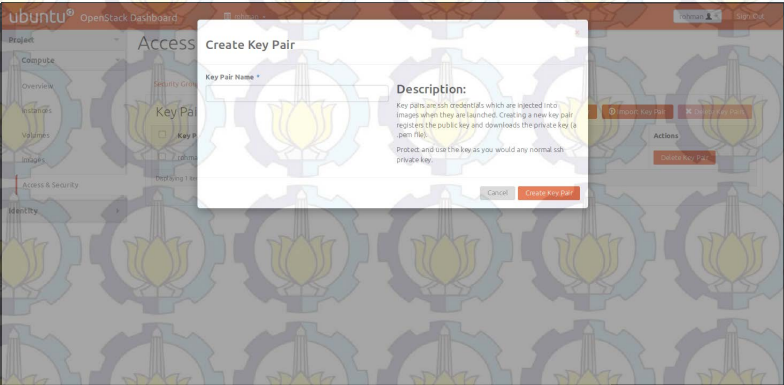
```
rohman@controller:~$ nova keypair-list
+-----+-----+
| Name   | Fingerprint |
+-----+-----+
| rohman-key | eb:82:f1:64:50:78:be:02:80:aa:db:cd:2e:f5:4b:d7 |
+-----+-----+
```

Gambar 4.13: Daftar *key pairs* melalui *console*

Penambahan *keypair* melalui antarmuka. Pengguna diharuskan memilih menu **Project**, kemudian **Compute** dan dilanjutkan dengan memilih menu **Access & Security**. Pada *tab* **Key Pairs** pilih **Create Key Pairs** dan masukan nama *key pair* seperti pada Gambar 4.14. Setelah proses penambahan selesai, maka akan muncul *key pairs* baru pada tabel *key pair* pada halaman tersebut, sebelum itu akan muncul sebuah halaman untuk mengunduh *key*

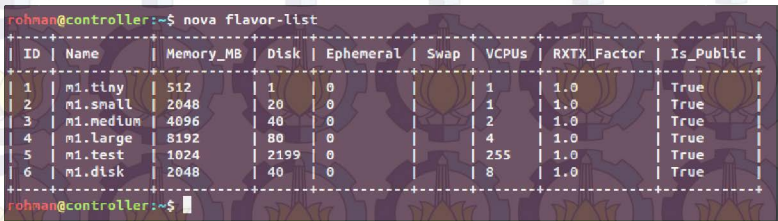


*pair* pada browser yang digunakan oleh pengguna. *Key pair* sangat penting, peran dari *key pair* sendiri seperti kunci yang digunakan untuk melakukan *remote* atau SSH (*Secure Shell*) ke mesin virtual yang akan dijelaskan pada bagian ini selanjutnya.



Gambar 4.14: Pembuatan *keypair* melalui antarmuka

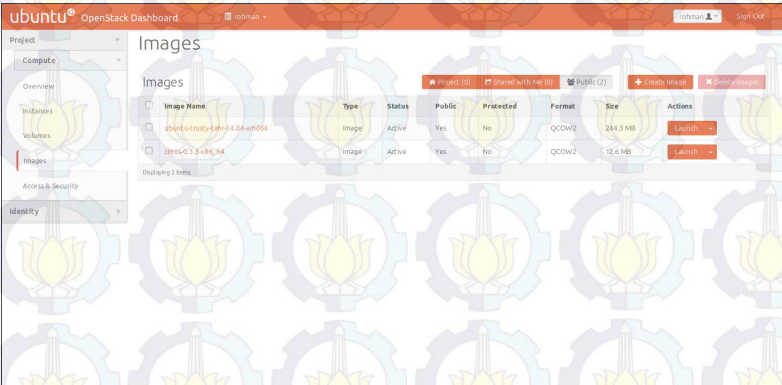
Selanjutnya pilih *flavor* yang disediakan oleh nova, *flavor* adalah pilihan paket spesifikasi perangkat keras yang akan digunakan oleh mesin virtual seperti yang terlihat pada Gambar 4.15. Pada pengujian ini paket yang dipilih adalah **m1.small**, dengan ukuran 1 GB untuk memori (RAM) dan 20 GB untuk penyimpanan. Melihat daftar *flavor* melalui antarmuka, pengguna dapat mengaksesnya pada saat menjalankan mesin virtual (*launch instance*) seperti pada Gambar 4.19.



Gambar 4.15: Daftar *flavor* melalui *console*

Pemilihan sistem operasi untuk mesin virtual ditunjukkan oleh Gambar 4.12 pada bagian uji coba nova. Di mana pengguna dapat

memilih satu sistem operasi untuk dipasang dari pilihan yang disediakan. Sedangkan untuk melihat daftar image melalui antarmuka, pengguna cukup memilih **Project**, kemudian **Compute** dan pilih **Image** seperti pada Gambar 4.16.



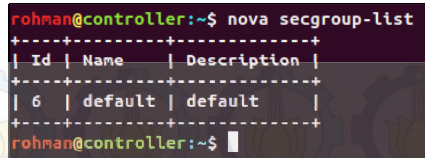
Gambar 4.16: Daftar *image* melalui antarmuka web

Nova juga menyediakan layanan jaringan bagi *instance*, setiap *instance* akan memperoleh alamat IP secara otomatis atau DHCP (*Dynamic Host Configuration Protocol*) melalui nova-network. Pada Gambar 4.17 menunjukan tentang alamat IP yang dialokasikan untuk seluruh pengguna komputasi awan dengan perintah `nova net-list`, tetapi perintah tersebut hanya bisa dijalankan oleh *admin* dengan script `admin-opensh.sh`.

```
rohman@controller:~$ nova net-list
+-----+-----+-----+
| ID | Label | CIDR |
+-----+-----+-----+
| 3e149db9-5f3e-4dfb-ac7b-23e07c4e2dfe | nova-net | 192.168.100.128/25 |
+-----+-----+-----+
rohman@controller:~$
```

Gambar 4.17: Daftar *nova-network* melalui *console*

Terdapat juga *secgroup* (*security group*) untuk menjaga keamanan *instance*. Pada penelitian ini *secgroup* yang digunakan adalah *secgroup* standar dari nova yang bekerja mirip dengan *firewall*. Gunakan perintah `nova secgroup-list` untuk melihat *secgroup* default seperti pada Gambar 4.18.



```

rohman@controller:~$ nova secgroup-list
+-----+-----+-----+
| Id | Name | Description |
+-----+-----+-----+
| 6 | default | default |
+-----+-----+-----+
rohman@controller:~$

```

Gambar 4.18: Daftar secgroup melalui *console*

Mengacu pada langkah-langkah sebelumnya untuk menjalankan *instance*, dengan masukan paket yang dipilih kemudian pilih sistem operasi yang akan digunakan, nomor ID (*Identification*) dari nova-network, security-group, security-key dan nama dari mesin virtual yang akan dibuat seperti pada Kode 4.7. Setelah perintah ini dijalankan maka akan tampil informasi dari mesin virtual yang telah sukses dibuat. Gunakan perintah *nova list* pada *console* untuk melihat status dari mesin virtual seperti pada Gambar 4.26. Saat mesin virtual galat atau berhasil dipasang juga dijelaskan pada tabel tersebut.

```

$ nova boot --flavor m1.small \
--image ubuntu-trusty-tahr-14.04-amd64 \
--nic net-id=3e149db9-5f3e-4dfb-ac7b-23e07c4e2dfe \
--security-group default --key-name rohman-key Ubuntu1

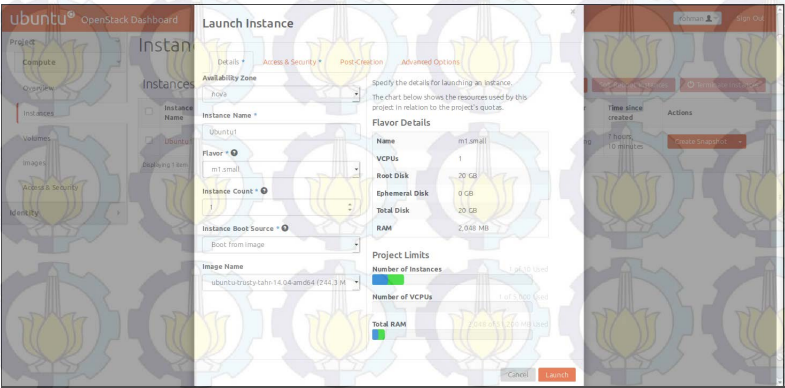
```

**Kode 4.7:** Perintah untuk membuat mesin virtual (*instance*) melalui *console*

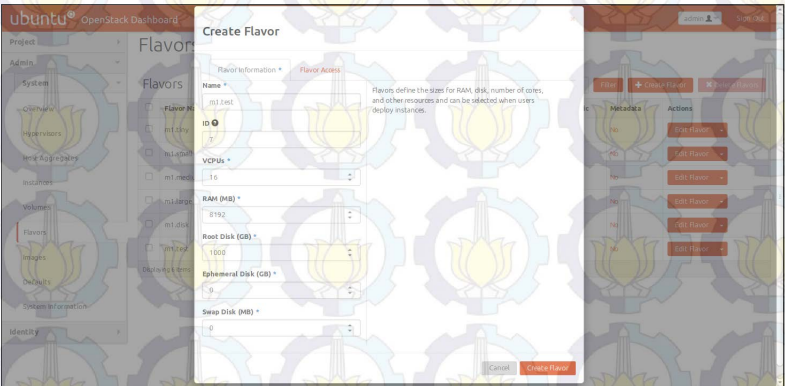
Menjalankan mesin virtual dengan antarmuka seperti pada Gambar 4.19 yang harus dilakukan adalah pilih **Project**, kemudian **Compute** dan dalam menu **Instance** pilih **Launch Instance**. Pada menu yang berupa *form* dan *dropdown*, isi dan sesuaikan dengan spesifikasi yang dibutuhkan. Contohnya nama “Ubuntu1”, selanjutnya pilih *flavor* sesuai dengan spesifikasi yang dibutuhkan misalkan **m1.small**. Kemudian masukan jumlah mesin virtual yang ingin dibuat pada *form* **Instance Count**. Pada **Instance Boot Source** pilih media yang digunakan untuk memasang sistem operasi. Misalkan **From Image**, maka nova akan terhubung dengan glance untuk membaca *image* dari sistem operasi yang disediakan.

Pengaturan spesifikasi untuk pembuatan mesin virtual dapat dilakukan dengan menambahkan *flavor* sesuai dengan kebutuhan

pengguna. Tetapi dalam penambahan *flavor* hanya bisa dilakukan oleh admin dengan mengisi *form* dan menu *dropdown* seperti pada Gambar 4.20. Setelah masuk sebagai admin, kemudian pilih menu **Admin**. Kemudian lanjutkan dengan memilih **Flavor** dan **Create Flavor**. Setelah *flavor* ditambahkan, maka *flavor* baru akan tampil pada halaman itu juga.



Gambar 4.19: Pembuatan mesin virtual melalui antarmuka

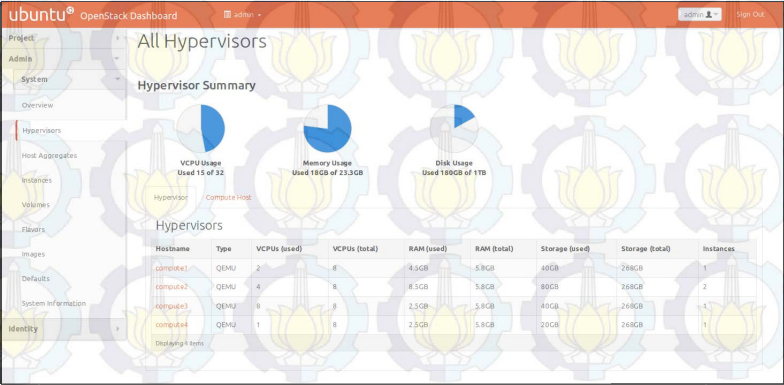


Gambar 4.20: Penambahan *flavor* melalui antarmuka

OpenStack memiliki berapa sistem monitoring untuk memantau kondisi layanan dan sumber daya pada menu **Admin**, con-



tohnya adalah **Overview**, **Hypervisors**, **Host Aggregates**, dan **System Information**. Memori, penyimpanan, dan CPU adalah contoh sumber daya yang dimonitorng secara keseluruhan agar *server* tetap bisa berjalan dengan baik. Pada Gambar 4.21 merupakan tampilan dari monitoring pada menu **hypervisors**, yang bertugas memonitor kondisi sumber daya pada setiap komputer server.



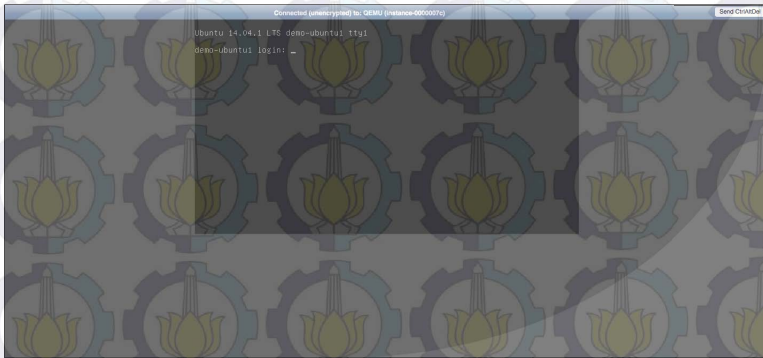
Gambar 4.21: Tampilan monitoring *hypervisors*

```
rohman@controller:~$ nova get-vnc-console Ubuntu1 novnc
+-----+
| Type | Url |
+-----+
| novnc | http://controller:6080/vnc_auto.html?token=b948a325-08e3-437a-865c-7ea6aae178fc |
+-----+
rohman@controller:~$
```

Gambar 4.22: Perintah tampilan alamat VNC yang berupa URL

Pengaksesan mesin virtual dapat dilakukan melalui banyak cara diantaranya menggunakan VNC (*Virtual Network Computing*). VNC memungkinkan pengguna dapat melakukan akses ke mesin virtual melalui *browser*. Pengujian pengaksesan mesin virtual dengan VNC dilakukan menggunakan dua cara, yaitu melalui *console* dan antarmuka web. Gunakan perintah `nova get-vnc-console Ubuntu1 novnc`, Ubuntu1 merupakan nama mesin virtual yang akan diakses seperti pada Gambar 4.22. Tampilah pada *console* alamat VNC dari mesin virtual yang berupa URL dan dapat diakses melalui *browser* seperti pada Gambar 4.23.

Penggunaan antarmuka *web* untuk mengakses VNC, pengguna dapat memilih menu **Project**. Kemudian pada menu **Compute** pilih **Instance**, pilih nama mesin virtual yang ingin diakses. Selanjutnya di dalam *tab console* pilih **Click here to show console only**, maka akan tampil sebuah *console* yang dapat dipakai melalui *browser*. Pada setiap pengaksesan alamat VNC yang berupa URL pada *browser*, terlebih dahulu rubah hostname *controller* dengan alamat IP dari *controller node*. Hal ini juga berlaku pada proses pengaksesan melalui *console* untuk memperoleh alamat VNC yang berupa URL.



**Gambar 4.23:** Akses mesin virtual melalui *browser*

Berbeda dengan pengaksesan mesin virtual dengan VNC yang dapat diakses melalui *browser*, dimana pengguna dapat memberikan perintah atau kontrol terhadap sistem operasi melalui antarmuka. Pengaksesan mesin virtual melalui *remote* atau SSH, pengguna harus mengatur konfigurasi dari *segroup*. Di mana *segroup* bekerja mirip seperti *firewall*, pengguna harus memberi izin agar mesin virtual dapat diakses dari luar jaringan sistem komputasi awan menggunakan *remote* atau SSH.

Pada Kode 4.8 merupakan perintah untuk memberi izin agar mesin virtual dapat diakses melalui SSH. Bukan hanya pengatur-an untuk pengaksesan melalui SSH, banyak protokol TCP/IP dan port lainnya dapat diatur. Misalnya ICMP (*Internet Control Mes-sage Protocol*) atau yang biasa disebut dengan PING, pada Ko-

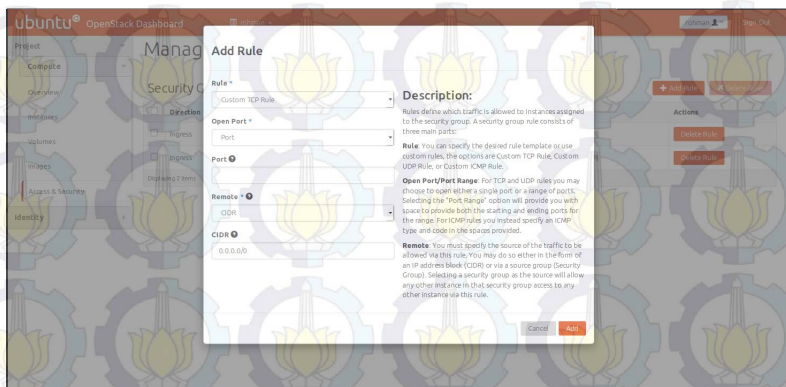
de 4.9 merupakan perintah untuk memberi izin agar mesin virtual dapat menerima paket ICMP dari luar. Pada perintah tersebut perintah yang digunakan adalah `nova secgroup-add-rule`, kemudian diikuti dengan nama *security group*, protokol dan *subnet* jaringan.

```
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

**Kode 4.8:** Perintah untuk memberikan izin agar mesin virtual dapat diakses melalui SSH

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
```

**Kode 4.9:** Perintah untuk memberikan izin agar mesin virtual dapat menerima paket ICMP

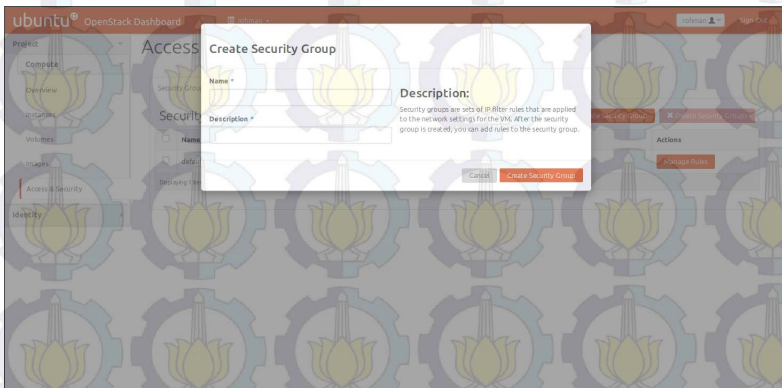


**Gambar 4.24:** Pengaturan *security group* melalui antarmuka

Pemberian izin agar mesin virtual dapat diakses melalui *remote* dapat dilakukan melalui antarmuka. Pengguna cukup memilih menu **Project**, kemudian dilanjutkan dengan **Compute** dan pilih menu **Access & Security**. Pada menu **Access & Security** dalam tab **Security Groups** terdapat sebuah tabel *security groups*. *Security groups* yang terdapat dalam tabel tersebut adalah **default**. Pemberian izin SSH, ICMP atau protokol TCP/IP lainnya dapat dilakukan dengan memilih menu **Manage Rules** dan dilanjutkan dengan **Add Rules**. Kemudian pengguna dapat memilih

SSH, ICMP atau protokol TCP/IP lainnya yang ingin dibuka atau diberi izin seperti Gambar 4.24.

Penambahan security group dapat dilakukan dengan memilih menu **Create Security Groups**, kemudian dilanjutkan dengan pengisian *form* yang disediakan seperti pada Gambar 4.25. Setelah penambahan selesai maka *security group* akan muncul pada tabelnya secara otomatis. Pengguna dapat menggunakannya saat pembuatan mesin virtual (*launch instance*), dan mengatur aturan perizinan setiap protokol seperti Gambar 4.24.



**Gambar 4.25:** Penambahan *security group* melalui antarmuka

#### 4.1.5 Uji Coba Cinder (Block Storage Service)

Pengujian pada cinder dilakukan dengan melakukan pengecekan apakah seluruh layanan yang mendasari cinder sudah terpasang dan tidak menunjukkan status galat. Pengecekan dilakukan dengan menggunakan perintah `cinder service-list` pada *console* seperti pada Gambar 4.27.



```

rohman@controller:~$ nova list
+-----+-----+-----+-----+-----+-----+-----+
| ID          | Name          | Status | Task State | Power State | Networks |
+-----+-----+-----+-----+-----+-----+
| 4d08cfe-3a99-42f1-bc96-7accd4c11cc3 | Ubuntu1       | ACTIVE | -           | Running     | nova-net=192.168.100.130 |
+-----+-----+-----+-----+-----+-----+
rohman@controller:~$

```

Gambar 4.26: Melihat daftar mesin melalui *console*

```

rohman@controller:~$ cinder service-list
+-----+-----+-----+-----+-----+-----+-----+
| Binary      | Host          | Zone   | Status | State | Updated at | Disabled | Reason |
+-----+-----+-----+-----+-----+-----+-----+
| cinder-scheduler | controller    | nova   | enabled | up    | 2015-05-10T16:04:30.000000 | None     | None   |
| cinder-volume    | controller    | nova   | enabled | up    | 2015-05-10T16:04:30.000000 | None     | None   |
+-----+-----+-----+-----+-----+-----+-----+
rohman@controller:~$

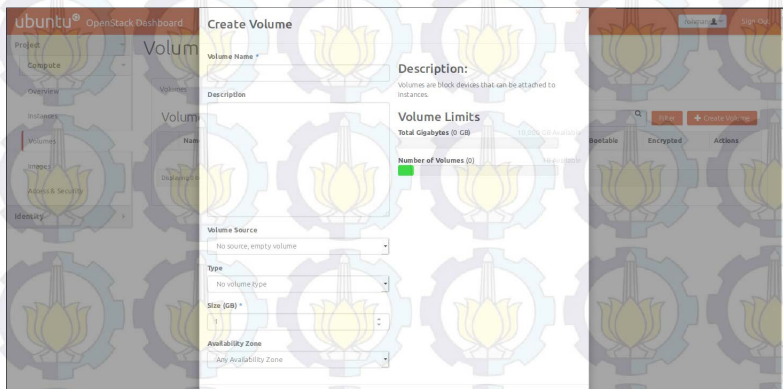
```

Gambar 4.27: Layanan-layanan pembangunan cinder

Di lanjutkan dengan pengujian dengan pembuatan penyimpanan (*volume*) tambahan untuk mesin virtual. Penggunaan perintah `cinder create` yang diikuti dengan penyimpanan dan ukurannya dengan satuan GB (*Gigabyte*) seperti pada Kode 4.10. Di mana nama dari penyimpanan yang dibuat adalah `Volume1` dengan kapasitas 1 GB.

```
$ cinder create --display-name Volume1 1
```

**Kode 4.10:** Perintah untuk menambah *volume* melalui *console*



**Gambar 4.28:** Penambahan *volume* melalui antarmuka

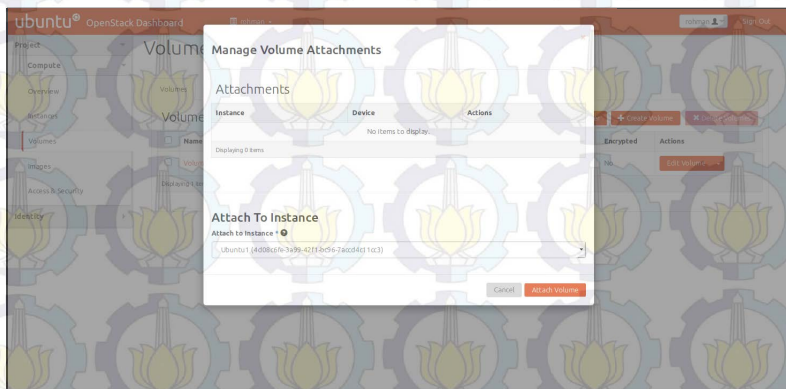
Pada penambahan *volume* melalui antarmuka, yang harus dilakukan pengguna adalah memilih **Project** dan dilanjutkan dengan **Compute**. Pilih **Volume** pada menu **Compute**, kemudian lanjutkan dengan **Create Volume** dan jangan lupa untuk mengisi *form* dan menu *dropdown* seperti pada Gambar 4.28. Pada pengujian ini, yang dilakukan adalah memasukkan nama *volume* dalam *form* nama penyimpanan dan *Size* untuk ukuran penyimpanan tambahan yang diinginkan. Proses tersebut sama dengan penggunaan perintah pada Kode 4.10

Penambahan penyimpanan yang telah dibuat ke mesin virtual juga dapat dilakukan dengan *console* dan juga antarmuka. Perintah pada *console* yang digunakan adalah `nova volume-attach`, yang

selanjutnya diikuti dengan nama mesin virtual dan nomor ID dari penyimpanan seperti pada Kode 4.11.

```
$ nova volume-attach Ubuntu1 \
158bea89-07db-4ac2-8115-66c0d6a4bb48
```

**Kode 4.11:** Perintah untuk menambah *volume* ke mesin virtual melalui *console*

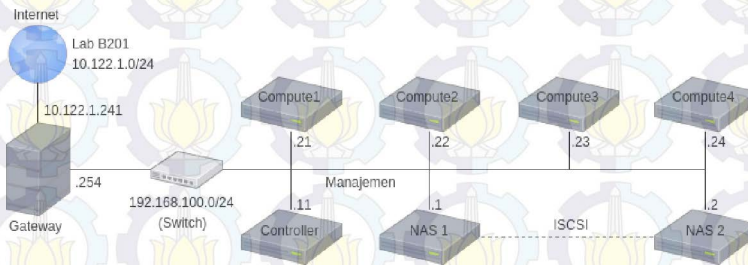


**Gambar 4.29:** Penambahan penyimpanan melalui antarmuka

Penambahan penyimpanan ke mesin virtual melalui antarmuka. Pengguna diharuskan untuk memilih **Project** kemudian **Compute**. Setelah memilih menu **Compute**, menu yang harus dipilih selanjutnya adalah **Volume**. Pada tabel penyimpanan terdapat sebuah kolom **Actions**, dan pada kolom tersebut terdapat *dropdown button*. Pada *dropdown button* kemudian pilih **Edit Attachments**, pilih mesin virtual yang ingin ditambah penyimpanannya pada menu *dropdown Attach to Instance* seperti pada Gambar 4.29. Status pada tabel penyimpanan akan berubah dan terdapat keterangan bahwa penyimpanan ditambahkan ke mesin virtual. Beracuan dengan proses penambahan penyimpanan ke mesin virtual melalui antarmuka dan cosole pada bagian ini, maka pengujian cinder dinyatakan berhasil.

## 4.2 Pengujian Topologi Jaringan

Terdapat dua jaringan yang terhubung pada *compute node* yaitu jaringan manajemen dan eksternal seperti pada Gambar 3.2. Pada awalnya jaringan yang digunakan hanya menggunakan manajemen, tanpa jaringan eksternal. Berawal hal tersebutlah pengujian dilakukan, penggunaan jaringan manajemen yang terhubung langsung melalui *gateway* seperti pada Gambar 4.30. Hasil dari penelitian tersebut gagal dan mesin virtual tidak bisa berjalan. Pada tabel daftar mesin virtual seperti pada Gambar 4.26, mesin virtual terus menunjukkan status *spawning* dalam jangka waktu lama dan kemudian berubah menjadi galat. Power state memberikan pesan *No State* dan alamat IP tidak muncul pada kolom *Networks Status*.



**Gambar 4.30:** Desain jaringan tanpa jaringan eksternal

Pada pengujian ini dapat ditarik kesimpulan bahwa mesin virtual pada OpenStack tidak dapat berjalan tanpa menggunakan jaringan eksternal. Di mana jaringan eksternal dari OpenStack terhubung dengan nova-network, yang berfungsi sebagai penyedia jaringan untuk mesin virtual yang berjalan pada setiap *compute node*.

## 4.3 Kemampuan Maksimal Komputasi Awan

Pengujian pada bagian ini bertujuan memperoleh data jumlah mesin virtual yang dapat dibuat oleh komputasi awan. Berapa batas spesifikasi yang dapat dicapai pada setiap mesin virtual juga dicari. Tiga elemen penting dari pengujian ini diantaranya komputasi (CPU), memory (RAM) dan penyimpanan. Ketiganya merupakan bagian penting dalam pembuatan mesin virtual pada sistem komputasi awan. Pada Tabel 3.1 dibagikan perangkat keras dan



sistem operasi menjelaskan spesifikasi dari *controller* dan *compute node*. Setiap komputasi pada sistem komputasi awan dijalankan pada *compute node*. Jumlah *compute node* yang digunakan sebanyak empat buah, ketika sumber daya dari setiap *node* dijumlah maka akan diperoleh spesifikasi seperti pada Tabel 4.1.

**Tabel 4.1:** Jumlah spesifikasi dari empat *compute node*

Spesifikasi	Keterangan
Jumlah Prosesor	8
Jumlah Core	32 core
Memori	24 GB
<i>Swap</i>	24 GB
<i>Disk Storage</i>	1200 GB

Setelah dilakukan pengujian batas maksimal dari komputasi awan, jumlah mesin virtual yang dapat dibuat bergantung pada jumlah memori. Semakin besar memori maka semakin banyak mesin virtual yang bisa dibuat, total memori yang dapat digunakan oleh pengguna memiliki batas perbandingan 1.5 : 1 dari jumlah memori pada setiap *compute node*. Kapasitas memori total dari *compute node* sebesar 24 GB, maka memori dari sistem komputasi awan sebesar 36 GB. Total memori sejumlah 36 GB tersebut diperoleh dari 24 GB memori dikali dengan 1.5. Pada komputasi jumlah maksimal core yang dapat dicapai sebesar 255 *core* dalam 1 mesin virtual, jumlah mesin virtual yang dapat dibuat menyesuaikan dengan jumlah memori. Sedangkan batas maksimal penyimpanan pada setiap mesin virtual adalah 2 GB.

#### 4.4 Perbandingan Performa Mesin Virtual

Pengujian bertujuan memperoleh data perbandingan performa antara mesin virtual layanan dari komputasi awan dengan sebuah komputer, keduanya menggunakan spesifikasi yang sama. Program *benchmark* yang digunakan adalah Phoronix Test Suite (PTS) dengan mode *interactive* dimana pengguna dapat memilih paket *benchmark* sesuai dengan kriteria yang ingin digunakan.

Seperti pada pengujian sebelumnya tiga elemen penting yang diukur dalam pengujian ini diantaranya komputasi (CPU), memory (RAM) dan penyimpanan (I/O). Selain itu dua pengujian lain juga turut ditambahkan pada bagian ini yaitu Apache Benchmark dan Compilation Benchmark. Kedua pengujian tersebut dilakukan karena terdapat hubungan antara CPU, memori dan penyimpanan dalam pemenuhan kebutuhan komputasi. Pada Tabel 4.2 merupakan spesifikasi (*flavor*) dari mesin virtual yang dibuat sama seperti spesifikasi komputer pada Tabel 4.3. Keduanya menggunakan sistem operasi Ubuntu 14.04 (Trusty Tahr) 64-bit.

**Tabel 4.2:** Spesifikasi (*flavor*) dari mesin virtual

Spesifikasi	Keterangan
Jumlah Prosesor	1
Jumlah Core	4 Core
Memori	8 GB
Swap	8 GB
<i>Disk Storage</i>	160 GB

**Tabel 4.3:** Spesifikasi dari komputer yang akan dibandingkan

Spesifikasi	Keterangan
Jumlah Prosesor	1
Jumlah Core	4 Core
Jenis Prosesor	AMD A8-3850 APU
Memori	8 GB
Swap	8 GB
<i>Disk Storage</i>	160 GB

Selain membandingkan performa antara mesin virtual yang merupakan layanan dari komputasi awan dengan sebuah komputer. Di bandingkan juga performa dari mesin virtual dengan spesifikasi yang terus dinaikan berdasarkan spesifikasinya. Pada pengujian ini yang diuji adalah performa dari memori dan komputasi (CPU)

saja. Pada Tabel 4.4 merupakan *flavor* atau spesifikasi dari mesin virtual dengan jumlah *core* yang terus dinaikan, hal ini bertujuan untuk menguji performa dari komputasi (CPU). Sedangkan pada Tabel 4.5 merupakan *flavor* atau spesifikasi dari mesin virtual dengan jumlah memori (RAM) yang terus dinaikan, hal ini bertujuan untuk menguji performa dari memori (RAM) dari mesin virtual. Melalui pengujian tersebut dapat terukur berapa kenaikan performa mesin virtual setiap kenaikan spesifikasinya.

**Tabel 4.4:** *Flavor* mesin virtual untuk mengukur performa komputasi.

Spec.	m3.test1	test2	test3	test4	test5
Processor	1	1	1	1	1
Core	1	2	4	8	16
Memori	2 GB	2 GB	2 GB	2 GB	2 GB
Swap	1 GB	1 GB	1 GB	1 GB	1 GB
Storage	160 GB	160 GB	160 GB	160 GB	160 GB

**Tabel 4.5:** *Flavor* mesin virtual untuk mengukur performa memori.

Spec.	test1	test2	test3	test4	test5
Processor	1	1	1	1	1
Core	4	4	4	4	4
Memori	512 MB	1 GB	2 GB	4 GB	8 GB
Swap	256 MB	512 MB	1 GB	2 GB	4 GB
Storage	160 GB	160 GB	160 GB	160 GB	160 GB

#### 4.4.1 Performa Komputasi (CPU)

Pengujian kinerja CPU secara keseluruhan dilakukan dengan mengukur kecepatan pemrosesan beban kerja. Melalui pengujian tersebut maka dapat ditarik sebuah standar perbandingan performa antara mesin virtual layanan dari komputasi awan dengan sebuah komputer. Tes profil dari Phoronix Test Suite yang digunakan dalam pengujian ini diantaranya C-ray, Stream, dan FFMpeg.

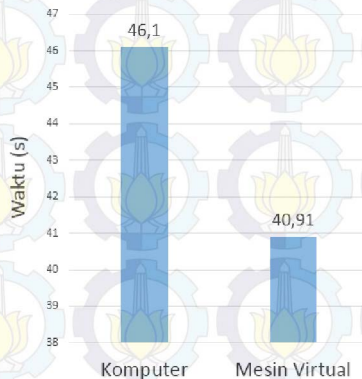
## C-Ray

C-Ray adalah sebuah program *raytracer* sederhana yang bertujuan untuk menguji kemampuan *floating-point* dari CPU. Pada tes ini akan dibuat 16 threads yang akan berjalan pada setiap *core* untuk menjalankan sebuah proses. Pada proses tersebut dilakukan 8 kali penembakan pada setiap piksel untuk proses *anti aliasing* (penghalusan gambar) dan akan dihasilkan sebuah gambar dengan ukuran 1600 x 1200 [7].

Tes ini hanya membutuhkan sedikit data dan tidak menggunakan RAM. Hanya CPU yang diuji dan hasilnya tidak dipengaruhi oleh komponen lain dari komputer atau mesin virtual. Hal tersebut yang menjadikan pengujian ini ideal untuk menguji kecepatan komputer.

**Tabel 4.6:** Waktu pemrosesan beban kerja antara mesin virtual dan komputer

Perangkat	Waktu (s)
Komputer	46,1
Mesin Virtual (VM)	40,91



**Gambar 4.31:** Perbedaan waktu pemrosesan beban kerja antara mesin virtual dengan komputer



Tolok ukur pada pengujian ini adalah waktu untuk memproses beban kerja. Semakin sedikit angka yang muncul pada hasil *benchmark* berarti semakin sedikit waktu yang dibutuhkan untuk memproses beban kerja. Pada Tabel 4.6 adalah hasil perbandingan antara mesin virtual dengan spesifikasi seperti pada Tabel 4.2 dan komputer yang spesifikasinya dijelaskan pada Tabel 4.3. Di mana kecepatan dari mesin virtual dalam memproses beban tersebut sekitar 40,91 detik, sedangkan komputer 46,10 detik. Hasil perbandingan tersebut juga dapat dilihat grafik perbandingan pada Gambar 4.31.

Semakin banyak penggunaan *thread* maka akan semakin mempercepat proses, tentunya penggunaan jumlah *thread* memiliki batas. Ketika penambahan *thread* terus dilakukan seperti pada pengujian ini, tetapi kecepatan mengolah beban sudah tidak bertambah. Maka dapat disimpulkan itu adalah kecepatan maksimal dari processor.

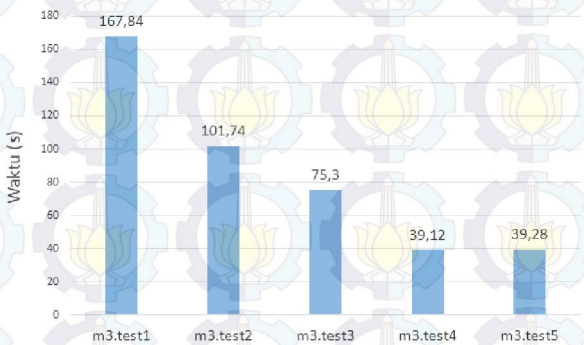
Selain itu diukur juga waktu dari mesin virtual dalam memproses beban kerja dengan spesifikasi yang terus dinaikan seperti pada Tabel 4.4 dengan menggunakan C-Ray. Spesifikasi yang digunakan seperti pada Tabel 4.4 dikarenakan fokus dari pengujian ini untuk mengukur kemampuan dari komputasi. Hasil dari pengujian ini dapat dilihat pada Tabel 4.7, dimana spesifikasi dari test1 yang kemudian ditambahkan sebuah *core* menjadi test2 terdapat pengurangan waktu sebesar 66,1 s. Sedangkan test2 ke test3 dimana terjadi penambahan *core* sejumlah 2 *core*, hal tersebut mempersingkat waktu pemrosesan sebesar 26,4 s. Pada test3 ke test4 terjadi penambahan *core* sejumlah 4 *core*, kemudian waktu pemrosesan berkurang lagi sebesar 36,16 s. Rata-rata waktu pemrosesan beban berkurang sebesar 18,41 s per-*core*.

Penambahan *core* pada test4 ke test5 sejumlah 8 *core*, tidak terlalu berpengaruh pada pemrosesan beban yang diuji dengan menggunakan C-Ray, perbandingan tersebut dapat dilihat pada Gambar 4.32. Kecepatan justru turun sebesar 18 s seperti pada Tabel 4.7, hal ini disebabkan oleh jumlah *core* dari *compute node* yang menjadi induk tempat berjalannya mesin virtual berjumlah lebih dari 8 *core*. Jika jumlah *core* melebihi dari 8 *core*, maka *core* tersebut merupakan *core* virtual dari OpenStack yang diperoleh

dari *compute node* lainnya. Performa *core* tambahan dari *compute node* lain tersebut berbeda jauh dengan performa dari *core* yang diperoleh dari *compute node* tempat mesin itu berjalan. Hal tersebut dikarenakan mesin virtual langsung melakukan akses ke perangkat keras melalui layer virtualisasi seperti dijelaskan pada bagian virtualisasi pada bab dasar teori.

**Tabel 4.7:** Waktu pemrosesan beban kerja pada mesin virtual

Flavor	Waktu (s)
test1	167,84
test2	101,74
test3	75,3
test4	39,12
test5	39,28



**Gambar 4.32:** Perbedaan waktu pemrosesan beban kerja pada mesin virtual

## Stream

*Stream* adalah sebuah patokan untuk mengukur memori *bandwidth*. Memori *bandwidth* dapat diartikan seberapa cepat data dapat ditulis atau dibaca dari memori oleh *processor*. Hal ini juga mempengaruhi seberapa cepat sistem operasi bisa memperoleh da-

ta dari memori untuk diproses. Jika bandwidth memori rendah, maka prosesor harus menunggu terlebih dahulu dalam setiap proses pengambilan dan penulisan data. Jika bandwidth memori tinggi, maka proses pengambilan dan penulisan data dapat dilakukan dengan cepat [8].

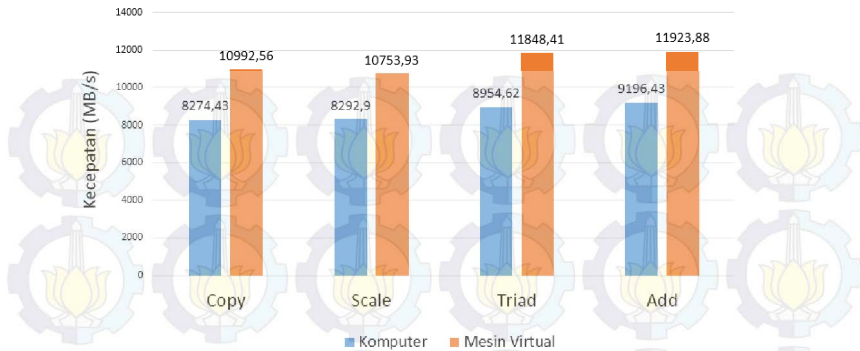
Pengukuran kecepatan dari stream terdiri dari empat kernel vektor sederhana yaitu *Copy*, *Scale*, *Add* dan *Triad*, satuan dari stream atau pengukuran memori bandwidth adalah MB/s [9]. Pada Tabel 4.8 adalah perbandingan hasil *stream benchmark*, semakin tinggi nilai benchmark maka semakin besar ukuran memori bandwidth. Pada komputer diperoleh kecepatan 8274,43 MB/s untuk *copy*, 8292,9 MB/s untuk *scale*, 8954,62 MB/s untuk *Triad* dan 9196,43 untuk *Add*. Sedangkan pada mesin virtual diperoleh kecepatan 10992,56 MB/s untuk *copy*, 10753,93 untuk *scale*, 11848,41 MB/s untuk *Triad* dan 11848,41 MB/s untuk *Add*. Hasil perbandingan tersebut juga dapat dilihat grafik perbandingan pada Gambar 4.33.

**Tabel 4.8:** Hasil pengujian memori *bandwidth* pada komputer dan mesin virtual

Perangkat	Add	Copy	Scale	Triad
Komputer	9196,43	8274,43	8292,9	8954,62
Mesin Virtual (VM)	11758,46	10992,56	10753,93	11923,88

Di ukur juga bandwidth dari mesin virtual dengan spesifikasi yang terus dinaikan seperti pada Tabel 4.4 dengan menggunakan Stream. Spesifikasi yang digunakan tetap seperti pada Tabel 4.4 dikarenakan fokus dari pengujian ini untuk mengukur memori bandwidth, seberapa cepat data dapat ditulis atau dibaca dari memori oleh *processor*.

Hasil dari pengujian ini dapat dilihat pada Tabel 4.16, dimana spesifikasi dari mesin virtual test1 yang kemudian ditambah jumlah *core* sebanyak 1 menjadi test2 terdapat penambahan kecepatan sebesar 1789,01 MB/s untuk proses *copy*, 1550,4 MB/s untuk *scale*, 902,33 MB/s untuk *triad* dan 871,73 MB/s untuk *add*. Sedangkan test2 ke test3 dimana terjadi penambahan *core* sejumlah 2 *core*,



**Gambar 4.33:** Perbandingan memori bandwidth antara mesin virtual dengan komputer

kecepatan mesin virtual bertambah sebesar 385,58 MB/s untuk proses *copy*, 682,27 MB/s untuk *scale*, 1584,79 MB/s untuk *triad* dan 1609,72 MB/s untuk *add*. Pada test3 ke test4 terjadi penambahan *core* sejumlah 4 *core*, kemudian kecepatan mesin virtual bertambah sebesar 936,72 MB/s untuk proses *copy*, 678,9 MB/s untuk *scale*, 665,66 MB/s untuk *triad* dan 615,03 MB/s untuk *add*. Rata-rata kecepatan dari mesin virtual bertambah sebesar 444,5 MB/s untuk proses *copy*, 415,93 MB/s untuk *scale*, 450,39 MB/s untuk *triad* dan 442,35 MB/s untuk *add* per-*core*.

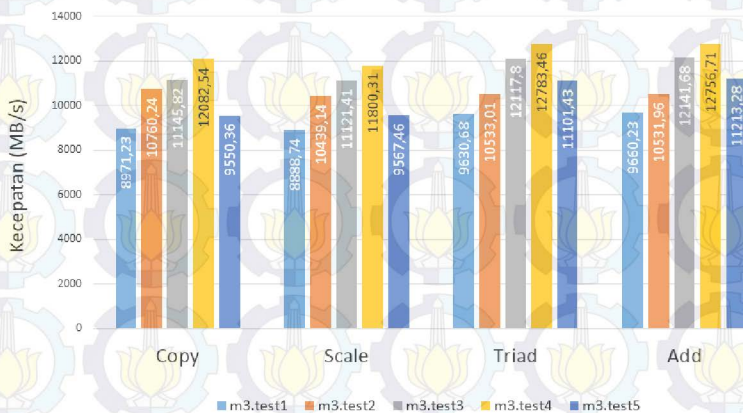
**Tabel 4.9:** Hasil pengujian performa memori *bandwidth* pada mesin virtual

Flavor	Add	Copy	Scale	Triad
test1	8971,23	8888,74	9630.68	9660.23
test2	11760.24	11439.14	12533.01	12531.96
test3	11145.82	11121.41	12117.8	12141.68
test4	12082.54	11800.31	12783.46	12756.71
test5	9550.36	9567.46	11101.43	11213.28

Penambahan *core* pada mesin virtual test4 ke test5 sejumlah 8 *core*, tidak terlalu berpengaruh pada pemrosesan beban yang diuji dengan menggunakan C-Ray, perbandingan tersebut dapat dilihat



pada Gambar 4.34. Kecepatan justru turun sebesar 2532,18 MB/s untuk proses *copy*, 2232,85 MB/s untuk *scale*, 1682,03 MB/s untuk *triad* dan 1543,43 MB/s untuk *add* seperti pada Tabel 4.16, hal ini disebabkan oleh jumlah core dari *compute node* yang menjadi induk tempat berjalannya mesin virtual berjumlah 8 core. Seperti penjelasan pada pengujian dengan menggunakan C-Ray. Jika core melebihi dari 8 core, maka core tersebut merupakan core virtual dari OpenStack yang diperoleh dari *compute node* lainnya. Performa *core* tambahan dari *compute node* lain tersebut berbeda jauh dengan performa dari *core* yang diperoleh dari *compute node* tempat mesin itu berjalan. Hal tersebut disebabkan mesin virtual langsung melakukan akses ke perangkat keras melalui layer virtualisasi seperti dijelaskan pada bagian virtualisasi pada bab dasar teori.



**Gambar 4.34:** Perbedaan memori bandwidth pada mesin virtual

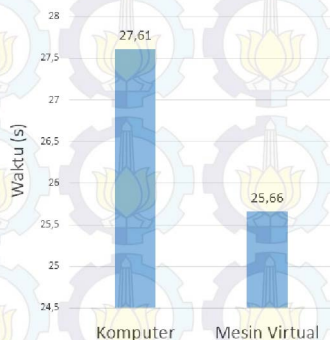
## FFmpeg

Pengujian performa *encoding* dengan menggunakan FFmpeg *encoding utility* yang disediakan oleh Phoronix Test Suite. Proses ini memakan RAM dan penyimpanan saat melakukan proses video *encoding*, tetapi saat melakukan pemrosesan *file* yang sangat besar, kunci utama dari proses ini adalah kemampuan CPU. Maka dari itu dibutuhkan kemampuan CPU yang cepat untuk konversi video [10].

Fakta tersebutlah yang mendasari perlunya pengujian ini dilakukan. Hasil perbandingan performa dapat dilihat pada Tabel 4.10, dimana kemampuan mesin virtual yang merupakan layanan dari komputasi awan mampu melakukan proses encoding dengan kecepatan 25,66 s. Sedangkan pada komputer, waktu yang dibutuhkananya adalah 27,61 *encoding*. Hasil perbandingan tersebut juga dapat dilihat grafik perbandingan pada Gambar 4.35.

**Tabel 4.10:** Hasil pengujian kecepatan proses encoding

Perangkat	Waktu (s)
Komputer	27,61
Mesin Virtual (VM)	25,66



**Gambar 4.35:** Perbandingan kecepatan encoding

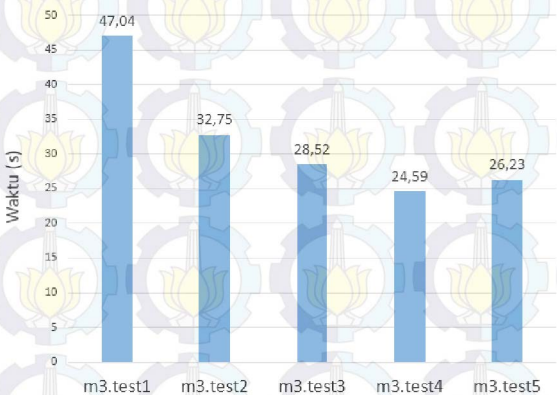
Selain itu diukur juga waktu dari mesin virtual dalam melakukan proses *encoding* dengan spesifikasi yang terus dinaikan seperti pada Tabel 4.4 dengan menggunakan FFmpeg *benchmark*. Hasil dari pengujian ini dapat dilihat pada Tabel 4.17, dimana spesifikasi dari test1 yang kemudian ditambahkan sebuah *core* menjadi test2 terdapat pengurangan waktu sebesar 14,29 s. Sedangkan test2 ke test3 dimana terjadi penambahan *core* sejumlah 2 *core*, hal tersebut mempersingkat waktu pemrosesan sebesar 4,23 s. Pada test3 ke test4 terjadi penambahan *core* sejumlah 4 *core*, kemudian wak-

tu pemrosesan berkurang lagi sejumlah 3,93 s. Rata-rata waktu pemrosesan beban berkurang sebesar 3,2 s per-core.

Penambahan *core* pada test4 ke test5 sejumlah 8 *core*, juga tidak terlalu berpengaruh pada proses *encoding* yang diuji dengan menggunakan FFmpeg *benchmark*, perbandingan tersebut dapat dilihat pada Gambar 4.41. Di mana waktu bertambah sebesar 1,64 s seperti pada Tabel 4.17, hal ini disebabkan oleh jumlah *core* dari *compute node* yang menjadi induk tempat berjalannya mesin virtual berjumlah lebih dari 8 *core*.

**Tabel 4.11:** Waktu yang diperlukan saat *encoding*

Flavor	Waktu (s)
test1	47,04
test2	32,75
test3	28,52
test4	24,59
test5	26,23



**Gambar 4.36:** Perbedaan waktu saat encoding

Seperti penjelasan pada pengujian dengan menggunakan C-Ray dan Stream. Jika jumlah *core* melebihi dari 8 *core*, maka *core* ter-

sebut merupakan core virtual dari OpenStack yang diperoleh dari *compute node* lainnya. Performa *core* tambahan dari *compute node* lain tersebut berbeda jauh dengan performa dari *core* yang diperoleh dari *compute node* tempat mesin itu berjalan. Hal tersebut dikarenakan mesin virtual langsung melakukan akses ke perangkat keras melalui layer virtualisasi seperti dijelaskan pada bagian virtualisasi pada bab dasar teori.

#### 4.4.2 Performa Penyimpanan

Penyimpanan bukan hanya dipengaruhi oleh besarnya kapasitas *hard disk drive* untuk menyimpan data. Performa dari penyimpanan tersebut juga berpengaruh dalam menangani setiap proses transaksi data, baik data itu diakses secara langsung oleh pengguna atau diakses secara otomatis oleh program. Tiga faktor yang berpengaruh pada performa penyimpanan diantaranya adalah *Throughput*, *IOPS (Input/Output Operations per Seconds)* dan *Latensi (Delay)*.

Maksud dari *Throughput* adalah berapa banyak data atau informasi yang dapat diakses dalam satuan waktu (MB/s), sedangkan *IOPS (Input/Output Operations per Seconds)* adalah masukan atau keluaran yang dapat dilakukan oleh penyimpanan dalam satuan detik. Kemudian untuk latensi secara umum adalah waktu saat melakukan permintaan pada komputer dan menerima jawabannya, sedangkan lebih spesifiknya pada penyimpanan dapat diartikan sebagai waktu pemilihan sektor untuk dibaca atau ditulis.

Tes profil yang digunakan dalam pengujian ini diantaranya *aio-stress*, *fio*, *dbench*, dan *postmark*. Berikut merupakan penjelasan tes dan perbandingan hasil *benchmark* pada penyimpanan.

##### Aio-stress

Tes ini memaksa penyimpanan untuk bekerja sesuai dengan kemampuan input dan outputnya secara tidak sinkron sampai batas kemampuannya. Tes ini menggunakan *file* berukuran 2048 MB yang terbagi menjadi 64 KB setiap bagiannya [7]. Setelah melakukan pengujian ini maka akan didapat *bandwidth* dari penyimpanan.

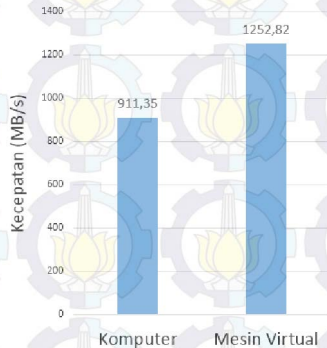
Hasil perbandingan performa dapat dilihat pada Tabel 4.7, dimana kemampuan mesin virtual yang merupakan layanan dari



komputasi awan memiliki *bandwidth* dan *throughput* lebih besar dari pada komputer. Di mana mesin virtual mampu memncapai kecepatan 1252,82 MB/s dan komputer mencapai 911,35 MB/s. Hasil perbandingan tersebut juga dapat dilihat grafik perbandingan pada Gambar 4.37.

**Tabel 4.12:** Hasil pengujian penyimpanan dengan Aio-Stress

Perangkat	Informasi per detik
Komputer	911,35
Mesin Virtual (VM)	1252,82



**Gambar 4.37:** Perbandingan bandwidth penyimpanan

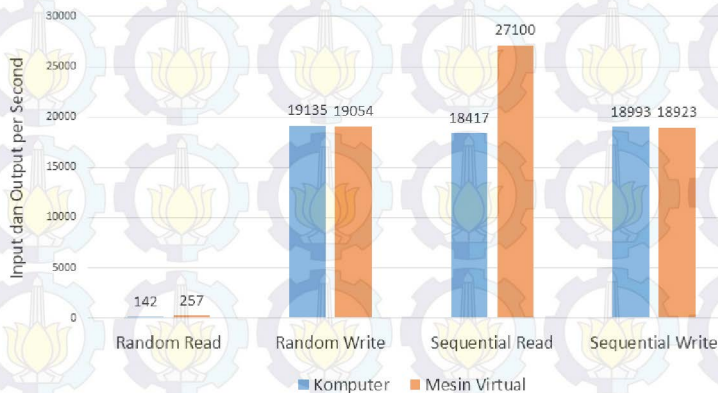
## Fio

Fio adalah kepanjangan dari *Flexible I/O Tester*. Tes ini akan memberikan beban kerja berupa input dan output pada penyimpanan. Pemberian beban tersebut berupa proses I/O yang dilakukan secara acak dan juga berurutan, selain itu proses pembacaan dan penulisan setiap beban juga turut di uji [10]. Tes ini menjalankan sebuah program yang mengakses penyimpanan dari mesin virtual dan komputer. Data-data yang diakses berukuran sangat kecil, rata-rata berukuran kurang dari 4 KB. Hal tersebut bertujuan mengukur kecepatan IOPS (*Input/Output Operations per Second*).

Hasil perbandingan performa dapat dilihat pada Gambar 4.38, menunjukkan kemampuan *input* atau *output* (I/O) dari mesin virtual dan komputer. Mesin virtual mampu menerima 275 *input* atau *output* melalui pengujian membaca secara acak (*random read*) dalam satu detik, sedangkan komputer hanya mampu menerima 142 I/O dalam setiap detiknya.

**Tabel 4.13:** Hasil pengujian penyimpanan dengan FIO

Perangkat	Random Read	Random Write	Sequential Read	Sequential Write
Komputer	142	19135	18417	18993
VM	257	19054	27100	18923



**Gambar 4.38:** Perbandingan I/O penyimpanan dengan FIO

Mengacu pada hasil pengujian pada Tabel 4.13, Di mana kemampuan penyimpanan dari mesin virtual dan komputer kembali diuji dengan tes penulisan secara acak (*random write*), mesin virtual memperoleh skor 19054 sedangkan komputer memperoleh skor 19135. Selain diuji dengan cara random atau acak, mesin virtual dan komputer diuji juga dengan cara yang urut. Pada pengujian membaca secara urut (*sequential read*) mesin virtual memperoleh nilai 27100, sedangkan komputer memperoleh nilai 18417. Di lan-

jutkan dengan pengujian penulisan secara urut (*sequential write*), maka diperoleh skor 18993 dari mesin virtual dan skor 18993 dari komputer. Perbandingan tersebut juga dapat dilihat pada Gambar 4.38. Nilai-nilai tersebut menunjukkan jumlah maksimal input atau output (I/O) yang bisa dilakukan dalam satuan detik.

#### 4.4.3 Performa Memori (RAM)

Sudah menjadi hal yang umum jika memori sangat berpengaruh pada pemrosesan beban kerja dan kecepatan dari komputer. Selain itu kemampuan RAM tidak hanya bergantung pada besarnya kapasitas memori, tetapi kecepatan memori, bandwidth dan cache juga memiliki pengaruh yang tidak kalah pentingnya.

Tes profil yang digunakan dalam pengujian ini diantaranya RAMspeed dan cachebench. Sedangkan pengujian pengukuran *bandwidth* pada memori sudah dilakukan pada bagian performa komputer (CPU) seperti yang ditampilkan pada Gambar 4.33.

##### RAMspeed

Hasil dari pengujian dengan menggunakan profil RAMspeed berupa *throughput* yang ditampilkan dalam satuan MB/s. Semakin tinggi angka hasil benchmark maka semakin baik performansinya. Terdapat dua nilai penting pada tes tersebut diantaranya *floating-point* dan *integer*. Selain itu terdapat juga empat metode pengujian yang berperan penting pada pengujian ini seperti add, copy, scale dan triad yang dijelaskan sebelumnya pada bagian performa komputer (CPU).

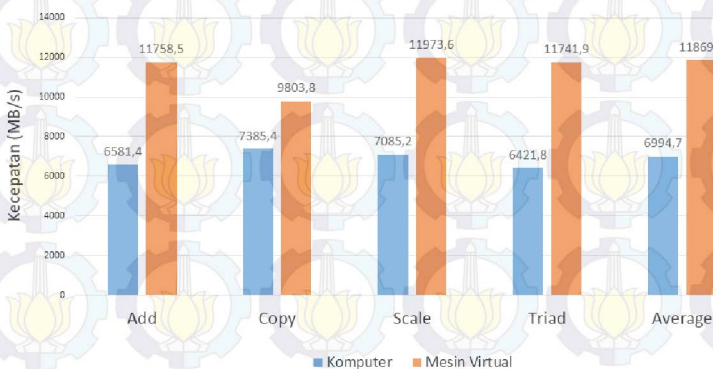
Pada Tabel 4.14 dapat dilihat perbandingan *throughput* antara mesin virtual dengan komputer saat memproses data integer dan pada Table 4.15 merupakan perbandingan saat memproses *floating-point*. Mesin virtual mampu mencapai 11869.96 MB/s untuk ngolah data integer dan memperoleh *throughput* 12104.3 MB/s untuk *floating point*. Sedangkan *throughput* dari komputer dapat mencapai 6994.73 MB/s untuk mengolah data *integer* dan 7631.58 MB/s untuk *floating point*. Perbandingan tersebut juga dapat dilihat pada Gambar 4.39 untuk pemrosesan data *integer* dan pada Gambar 4.39 untuk *floating point*.

**Tabel 4.14:** Hasil pengujian *throughput* memori saat mengolah data *integer* pada mesin virtual dan komputer

Perangkat	Add	Copy	Scale	Triad	Average
Komputer	6581,44	7385,36	7085,23	6421,75	6994,73
VM	11758,46	9803,82	11973,57	11741,93	11869,96

**Tabel 4.15:** Hasil pengujian *throughput* memori saat mengolah *floating-point* pada mesin virtual dan komputer

Perangkat	Add	Copy	Scale	Triad	Average
Komputer	7602,5	7366,8	7161,3	7835,6	7631,6
VM	12825,1	11527	11548,7	12799,8	12104,3



**Gambar 4.39:** Perbandingan *throughput* saat memproses *integer* pada komputer dan mesin virtual

Di ukur juga kecepatan memori dari mesin virtual dengan spesifikasi yang terus dinaikan seperti pada Tabel 4.5 dengan menggunakan RAMspeed. Spesifikasi yang digunakan tetap seperti pada Tabel 4.5 dikarenakan fokus dari pengujian ini untuk mengukur *throughput* dari memori, seberapa cepat memori dapat memproses data *integer* dan *floating-point*. Hasil dari pengujian ini dapat dilihat pada Tabel 4.16 yang merupakan perbandingan *throughput*



pada mesin virtual saat mengolah data integer dan pada Tabel 4.17 adalah saat pemrosesan data *floating-point*. Keduanya dibandingkan dengan cara menaikkan jumlah memorinya, kemudian ditampilkan perbandingannya pada Gambar 4.41 dan Gambar 4.42.

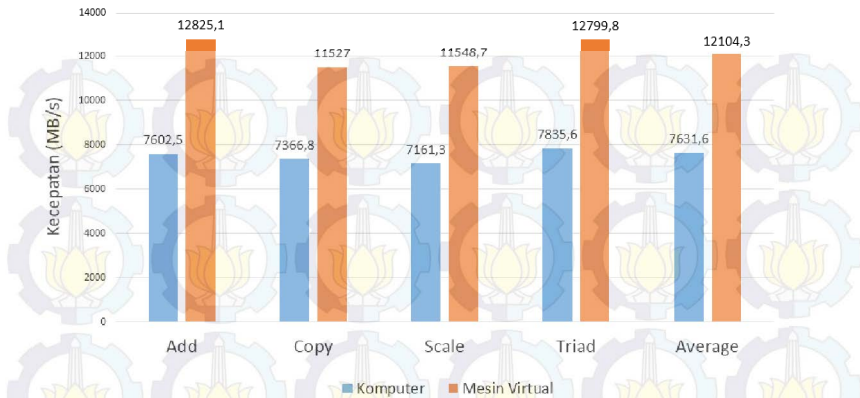
**Tabel 4.16:** Hasil pengujian *throughput* saat memproses data *integer* pada mesin virtual

Flavor	Add	Copy	Scale	Triad	Average
test1	4295,32	2992,39	2893,51	2910,12	2906,54
test2	3548,67	3072,18	2952,02	2956,76	2993,99
test3	3061,82	2982,32	3013,37	2986,03	3025,12
test4	3096,77	2975,57	2977,99	2990,69	3132,68
test5	11169,13	11564,66	11629,49	11042,92	11391,08

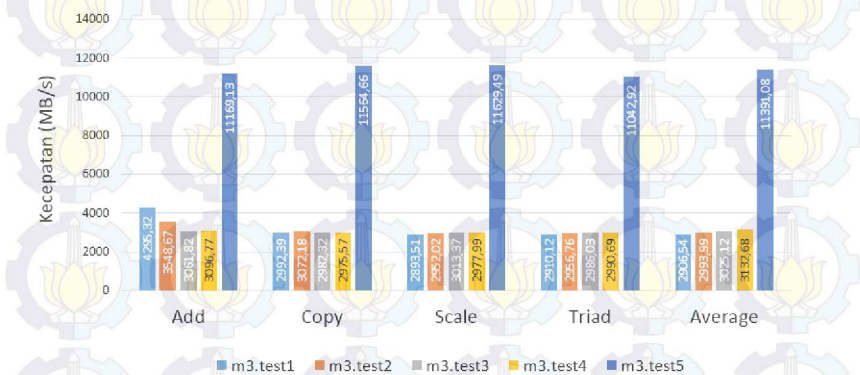
**Tabel 4.17:** Hasil pengujian *throughput* saat memproses *floating-point* pada mesin virtual

Flavor	Add	Copy	Scale	Triad	Average
test1	3090,08	2989,48	2907,53	3328,14	3223,74
test2	3263,81	2968,35	3082,33	3357,2	3649,7
test3	3453.79	2980.2	2989.58	3382,68	4583,37
test4	3333.98	2991.91	3142.49	3257.95	4441.79
test5	12291.92	11357.5	10072.47	12315.75	11892.81

Data perbandingan diperoleh dari penambahan spesifikasi dari mesin virtual test1 yang kemudian ditambah jumlah memorinya sebesar 512 MB menjadi test2 terdapat penambahan kecepatan sebesar 87.4 MB/s dalam mengolah data *integer* dan 425.96 MB/s untuk 425.96 MB/s untuk *floating-point*. Sedangkan test2 ke test3 dimana terjadi menambahkan memori sebesar 1 GB, kecepatan mesin virtual bertambah sebesar 31.13 MB/s untuk *integer*, dan 933.67 MB/s untuk *floating-point*.



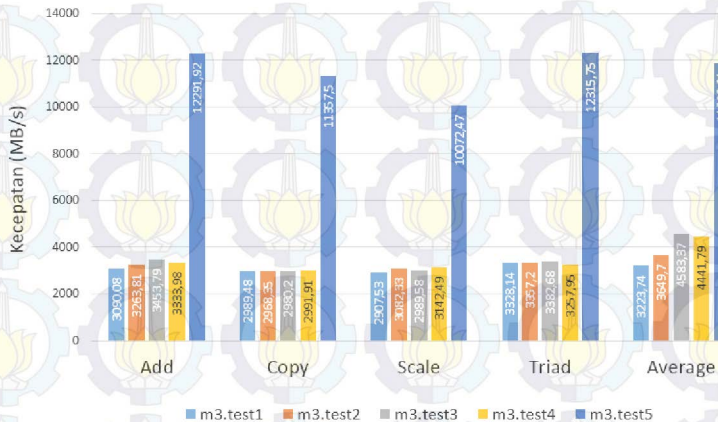
**Gambar 4.40:** Perbandingan *throughput* saat memproses *floating point* antara mesin virtual dengan komputer



**Gambar 4.41:** Perbedaan *throughput* saat memproses data *integer* pada mesin virtual.

Pada test3 ke test4 terjadi penambahan memori sebesar 2 GB, kemudian kecepatan mesin virtual bertambah sebesar 107.56 MB/s untuk integer dan sedangkan untuk floating-point mengalami penurunan kecepatan sebesar 141.58 MB/s dari pengujian sebelumnya pada *flavor* test test3. Sedangkan pada test4 ke test5 dimana terjadi penambahan memori sebesar 4 GB, kecepatan mesin virtual ber-

tambah sebesar 8258.4 MB/s untuk pemrosesan data *integer*, dan 7451.02 MB/s untuk *floating-point*. Rata-rata kecepatan dari mesin virtual bertambah sebesar 1205,83 MB/s untuk pemrosesan data *integer* dan 1208,01 MB/s untuk pemrosesan *floating-point* setiap penambahan 1 GB memori.



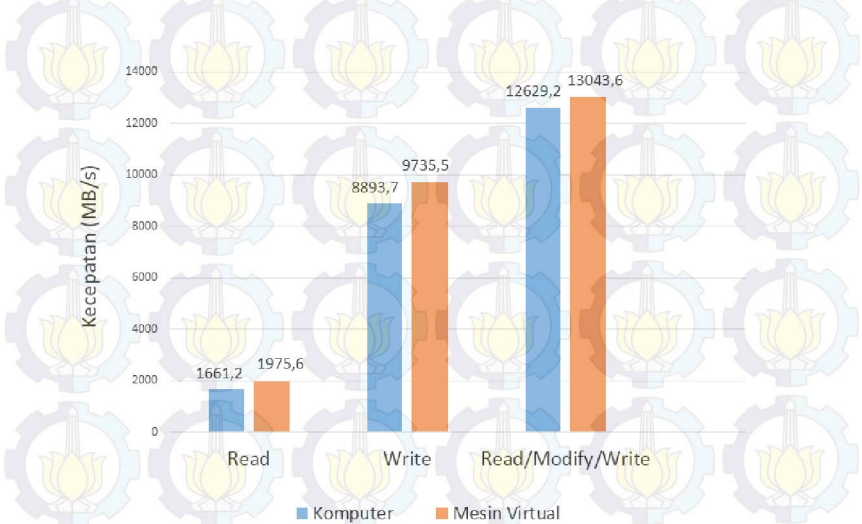
**Gambar 4.42:** Perbedaan *throughput* saat memproses *floating-point* pada mesin virtual

## Cachebench

CacheBench dirancang untuk menguji performa memori dan bandwidth cache [7]. Pada Tabel 4.18 merupakan hasil perbandingan performa memori cache yang terdapat pada mesin virtual dan komputer. Di mana saat proses membaca (*read*) mesin virtual dapat mencapai kecepatan 1975.64 MB/s. Kemudian untuk menulis (*write*) mesin virtual mampu mencapai 9735.51 MB/s dan untuk proses Read/Modify/Write dapat mencapai 13043.6 MB/s seperti pada Tabel 4.18. Pada komputer proses membaca dapat mencapai kecepatan 1661.15 MB/s, 8893.67 MB/s untuk proses menulis dan 12629.21 MB/s untuk proses *read/write/modify*. Hasil perbandingan tersebut juga dapat dilihat grafik perbandingan pada Gambar 4.43.

**Tabel 4.18:** Hasil perbandingan kecepatan *cache* memori antara mesin virtual dengan komputer

Perangkat	Read	Write	Read/Modify/Write
Komputer	277	35153	18947
Mesin Virtual (VM)	917	19054	27327



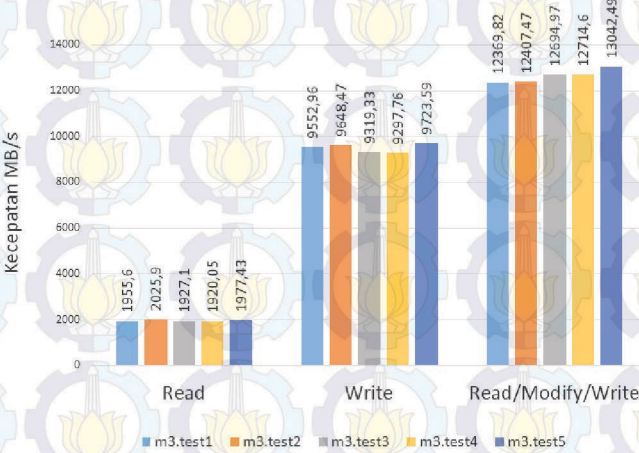
**Gambar 4.43:** Perbandingan kecepatan *cache* memori antara mesin virtual dengan komputer

Selain itu diukur juga kecepatan *cache* memori dari mesin virtual dengan spesifikasi yang terus dinaikan seperti pada Tabel 4.5 dengan menggunakan CacheBench. Spesifikasi yang digunakan tetap seperti pada Tabel 4.5 dikarenakan fokus dari pengujian ini untuk mengukur bandwidth dari *cache* memori, seberapa cepat *cache* memori dalam melakukan proses baca, tulis dan *read/write/modify*. Hasil dari pengujian ini dapat dilihat pada Tabel 4.17 yang merupakan perbandingan *bandwidth* pada mesin virtual saat menjalankan proses baca, tulis dan *read/write/modify*. Keduanya dibandingkan dengan cara menaikkan jumlah memorinya, hasil perbandingannya dapat dilihat pada Gambar 4.44.



**Tabel 4.19:** Hasil perbandingan *bandwidth cache* memori pada mesin virtual

Flavor	Read	Write	Read/Modify/Write
test1	1955,6	9552,96	12369,82
test2	2025,9	9648,47	12407,47
test3	1927,1	9319,33	12694,97
test4	1920,05	9297,76	12714,6
test2	1977,43	9723,59	13042,49



**Gambar 4.44:** Perbedaan *bandwidth cache* memori mesin virtual

Data tersebut diperoleh dari penambahan spesifikasi dari mesin virtual test1 yang kemudian ditambah jumlah memorinya sebesar 512 MB menjadi test2 terdapat penambahan kecepatan sebesar 70.3 MB/s saat proses membaca, 95.51 MB/s saat menulis dan 37.65 MB/s saat proses *read/write/modify*. Sedangkan test2 ke test3 dimana terjadi penurunan kecepatan sebesar 98.8 MB/s saat proses membaca, penurunan sebesar 329.14 MB/s juga terjadi saat menulis, akan tetapi terjadi kenaikan sebesar 287.5 MB/s saat proses *read/write/modify*.

Pada test3 ke test4 dilakukan penambahan memori sebesar 2 GB, kemudian kecepatan mesin virtual justru semakin berkurang sebesar 7.05 MB/s saat proses membaca, berkurang sebesar 21.56 MB/s saat menulis, akan tetapi bertambah 19.63 saat proses *read/write/modify*. Sedangkan pada test4 ke test5 dimana dilakukan penambahan memori sebesar 4 GB, kecepatan mesin virtual bertambah sebesar 57.38 MB/s untuk proses membaca, bertambah sebesar 425.83 MB/s untuk proses menulis dan bertambah sebesar 327.88 MB/s saat proses *read/write/modify*. Rata-rata kecepatan dari mesin virtual berkurang sebesar 1,9 MB/s untuk proses baca, bertambah sebesar 38,02 MB/s saat proses tulis dan bertambah sebesar 93,4 MB/s saat proses *read/write/modify* setiap penambahan 1 GB memori.

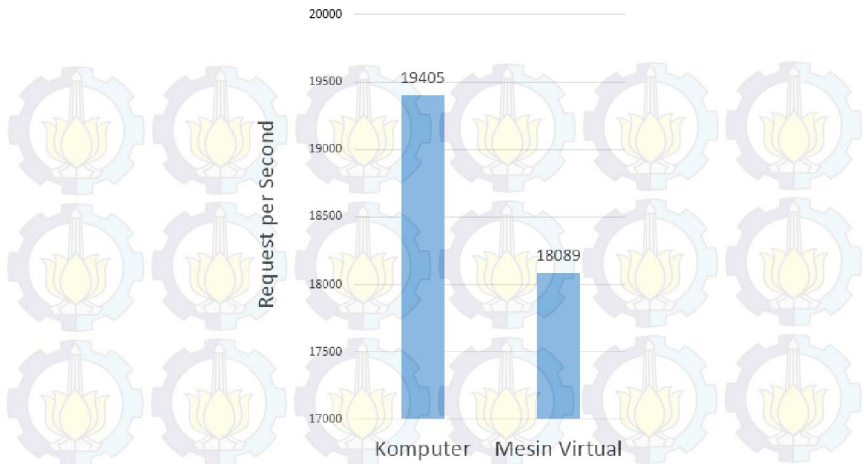
#### 4.4.4 Apache Benchmark

Pada pengujian ini, komputer dan mesin virtual diberi tugas sebagai *web server*. Beban yang akan diberikan pada komputer atau mesin virtual berupa permintaan sejumlah 100 sampai dengan 1.000.000 setiap detiknya [7]. Di desain untuk memberikan gambaran performa dari *webserver*, secara khusus akan menampilkan seberapa banyak request per detik atau seberapa banyak request per detik yang bisa dilayani mesin virtual atau komputer.

Pada Tabel 4.20 merupakan perbandingan performa antara mesin virtual dengan komputer saat berperan sebagai *webserver*. Komputer mampu menangani sekitar 19405 permintaan dalam setiap detiknya, sedangkan mesin virtual bisa menangani sekitar 18089 permintaan. Hasil perbandingan tersebut juga dapat dilihat grafik perbandingan pada Gambar 4.45.

**Tabel 4.20:** Hasil pengujian mesin virtual dan komputer sebagai *web-server* dengan Apache Benchmark

Perangkat	Permintaan per Detik
Komputer	19405
Mesin Virtual (VM)	18089



**Gambar 4.45:** Kemampuan menerima permintaan setiap detiknya

## DAFTAR PUSTAKA

- [1] M. Armbrust, A. Fox, R. Griffith, et al., Above the cloud : A Berkeley View of Cloud Computing. No. UCB/EECS-2009-28, USA: EECS Department, University of California at Berkeley, February 2009. (Dikutip pada halaman 1).
- [2] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, "Comparison of open-source cloud management platforms: Openstack and opennebula," in Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on, pp. 2457–2461, May 2012. (Dikutip pada halaman 1, 7).
- [3] B. Dordevic, S. Jovanovic, and V. Timcenko, "Cloud computing in amazon and microsoft azure platforms: Performance and service comparison," in Telecommunications Forum Telfor (TELFOR), 2014 22nd, pp. 931–934, Nov 2014. (Dikutip pada halaman 1).
- [4] "Openstack installation guide for ubuntu 14.04." <http://docs.openstack.org/>. Terakhir diakses pada tanggal 18 Mei 2015. (Dikutip pada halaman 5, 7, 16).
- [5] M. Mahjoub, A. Mdhaftar, R. Halima, and M. Jmaiel, "A comparative study of the current cloud computing technologies and offers," in Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on, pp. 131–134, Nov 2011. (Dikutip pada halaman 8).
- [6] W. Zhang, H. Xie, and R. Hsu, "Automatic memory control of multiple virtual machines on a consolidated server," Cloud Computing, IEEE Transactions on, vol. PP, no. 99, pp. 1–1, 2015. (Dikutip pada halaman 9).
- [7] "Phoronix test suite tests." <http://openbenchmarking.org/tests/pts>. Terakhir diakses pada tanggal 31 Mei 2015. (Dikutip pada halaman 54, 62, 69, 72).
- [8] J. Layton, "Finding memory bottlenecks with stream." <http://www.admin-magazine.com/HPC/Articles/>



Finding-Memory-Bottlenecks-with-Stream. Terakhir diakses pada tanggal 31 Mei 2015. (Dikutip pada halaman 57).

- [9] K. Raman, "Optimizing memory bandwidth on stream triad." <https://software.intel.com/en-us/articles/optimizing-memory-bandwidth-on-stream-triad>.

Terakhir diakses pada tanggal 31 Mei 2015. (Dikutip pada halaman 57).

- [10] "Cloud servers benchmarks." <http://www.sherweb.com/blog/>. Terakhir diakses pada tanggal 31 Mei 2015. (Dikutip pada halaman 59, 63).

## BAB 5

### PENUTUP

#### 5.1 Kesimpulan

Dari hasil implementasi dan pengujian mesin virtual pada komputasi awan yang sudah dilakukan dapat ditarik beberapa kesimpulan sebagai berikut :

1. Di perlukannya penggunaan jaringan eksternal pada setiap *compute node*, hal tersebut diperlukan oleh nova-compute dalam melakukan pembagian jaringan dan alamat IP bagi setiap mesin virtual. Pada pengujian jaringan, jaringan eksternal digabung menjadi satu dengan jaringan manajemen. Hasilnya mesin virtual dapat dibuat, tetapi tidak berjalan dan tidak memperoleh alokasi alamat IP.
2. Hasil pengujian batas maksimal mesin virtual, jumlah mesin virtual yang dapat dijalankan bergantung pada jumlah memori (RAM) komputer induk. Semakin besar memori dari komputer induk maka semakin besar jumlah mesin virtual yang bisa dibuat, total dari memori yang dapat digunakan oleh pengguna memiliki batas perbandingan 1.5 : 1 dari memori pada komputer induk.
3. Setiap peningkatan 1 core pada CPU dapat berpengaruh pada berkurangnya waktu pemrosesan beban sebesar 18 s. Sedangkan pada *bandwidth* memori atau kecepatan membaca dan menulis data pada memori oleh *processor*, diperoleh rata-rata peningkatan kecepatan mesin virtual sebesar 444,5 MB/s untuk proses *copy*, 415,93 MB/s untuk *scale*, 450,39 MB/s untuk *triad* dan 442,35 MB/s untuk *add*. Selain itu waktu *encoding* juga berkurang sebesar 1,64 s, artinya proses tersebut semakin cepat.
4. Setiap penambahan 1 GB memori mampu mempengaruhi *throughput* rata-rata mesin virtual sebesar 1205,83 MB/s untuk pemrosesan data *integer* dan 1208,01 MB/s untuk pemrosesan *floating-point*, dimana rata-rata tersebut diperoleh melalui proses *add*, *copy*, *scale* dan *triad*. Sedangkan pada *cache* me-

mori mesin virtual, bandwidth justru berkurang sebesar 1,9 MB/s untuk proses baca, bertambah sebesar 38,02 MB/s saat proses tulis dan bertambah lagi sebesar 93,4 MB/s saat proses *read/write/modify* setiap penambahannya.

## 5.2 Saran

Demi pengembangan lebih lanjut mengenai tugas akhir ini, disarankan beberapa langkah lanjutan sebagai berikut :

1. Penambahan alamat IP publik supaya komputasi awan dapat diakses dari luar dan penelitian dapat dilakukan lebih luas.
2. Penambahan *Networking node* supaya pengaturan jaringan beserta sistem keamanan pada komputasi awan lebih tertata dan memiliki perangkat pengatur jaringan pada komputasi awan yang lebih spesifik.
3. Dilakukan penambahan memori atau komputer server baru agar dapat membuat lebih banyak mesin virtual untuk menangani kebutuhan komputasi yang beragam di Lab B201.